MARIUS TOMESCU CARMEN FIFOR

Programarea calculatoarelor

Introducere în C#

Editura Universităţii "Aurel Vlaicu" Arad
- 2010 -

Prefaţă

C# este un limbaj de programare de nivel înalt, orientat obiect, care face parte din tehnologia software .Net, şi reprezintă un pas important în evoluţia limbajelor de programare. Multe firme recurg la limbajul C# în procesele de implementare şi dezvoltare software deoarece oferă un foarte bun suport pentru tehnologiile de vârf.

Cursul de faţă cuprinde atât noţiunile de bază ale limbajului C#, cât şi unele elemente de nivel mediu şi avansat. Cursul poate fi un instrument util pentru însuşirea şi consolidarea cunoştinţelor pentru oricine doreşte să înţeleagă şi să fie iniţiat într-un limbaj de programare. Primele şase capitole cuprind noţiuni de bază ale limbajului C#: tipuri de date şi instrucţiuni elementare, care au rolul de a iniţia studentul în limbajul de programare C# aducându-l la un nivel mediu, noţiuni introductive despre clase şi folosirea fişierelor. Ultimul capitol (PROBLEME REZOLVATE) are un rol important în pregătire. În acest capitol sunt prezentate şi explicate în detaliu programe de nivel mediu spre complex, cu scopul de a clarifica noţiunile fundamentale de programare C#, explicaţii menite a forma o gândire algoritmică riguroasă.

Sperăm ca textul acestui curs să ofere o introducere plăcută în programarea calculatoarelor şi limbajul C#. Am încercat să prezentăm fiecare noţiune într-o manieră accesibilă şi interesantă.

Scopul acestui curs este de a va învăţa noţiunile de bază ale programării în C# şi de realiza trecerea studentului de la un cunoscător al limbajului C# la un programator C#.

Cursul se adresează în primul rând studenţilor din anul I, celor care studiază un curs elementar sau academic de algoritmi, structuri de date şi limbaje de programare.

Octombrie 2010 *Autorii*

CUPRINS

CAPITOLUL I. INTRODUCERE ÎN LIMBAJUL C#

1.1 Conceptele de bază ale limbajului C#

Fiecare nou limbaj de programare este influenţat într-o anumită măsură de limbaje de programare deja existente. C# derivă direct din două dintre cele mai de succes limbaje de programare pe plan mondial şi anume C (inventat de către Dennis Ritchie în anii ’70, rulând iniţial pe sistemul de operare Unix) şi C++ (inventat de către Bjarne Stroustrup în 1979 pe baza limbajului C, noutatea fundamentală fiind introducerea programării orientate spre obiecte). De asemenea, se înrudeşte cu un alt limbaj mai nou şi deopotrivă de succes şi anume limbajul Java (dezvoltat de către firma Sun Microsystems începând cu anul 1991, având o sintaxă şi o filozofie derivate din C++, contribuţia fundamentală fiind portabilitatea programelor, necesară odată cu dezvoltarea diverselor sisteme de operare şi a Internetului).

Chiar dacă Java a rezolvat cu succes multe din problemele legate de portabilitate în Internet, C# a venit să rezolve una din facilităţile care îi lipseau şi anume *interoperabilitatea limbajelor de programare diferite*, denumită şi *programare în limbaj mixt*. Aceasta reprezintă posibilitatea codului scris într-un anumit limbaj să lucreze împreună cu codul scris într-un alt limbaj, necesară în special la crearea sistemelor software distribuite, de mari dimensiuni.

C# este un limbaj de programare orientat-obiect conceput de Microsoft la sfârşitul anilor ’90. Acesta fost conceput ca un concurent pentru limbajul Java. C# este un derivat al limbajului de programare C++ şi a fost dezvoltat de o echipă restrânsă de ingineri de la Microsoft, echipă din care s-a evidenţiat Anders Hejlsberg (autorul limbajului Turbo Pascal şi membru al echipei care a proiectat Borland Delphi). Limbajul C# este simplu, cu circa 80 de cuvinte cheie, şi 12 tipuri de date predefinite. El permite programarea structurată, modulară şi orientată obiect, conform preceptelor moderne ale programării profesioniste.

Limbajul de programare C# a fost proiectat pe baza experienţelor acumulate din celelalte limbaje de programare. Întregul limbaj C# se bazează pe conceptul de *obiecte*. În esenţă, structura unui program scris în C# este gândită pe mai multe niveluri, aşa cum se arată în figura următoare.

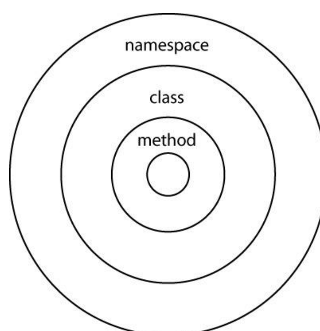


Figura 1. Într-un program C# vom avea cod de program în care se definesc *metode,* care sunt incluse în *clase*, care la rândul lor sunt incluse în *nume de spaţii*.

1.2 Structura unui program C#

Un program scris într-un limbaj de programare constă din instrucţiuni în limba engleză, denumite *cod sursă*. O instrucţiune a unui limbaj de programare reprezintă o comandă dată

calculatorului. Totalitatea instrucţiunilor care descriu cum se rezolvă o anumită problemă se numeşte *program*. Structura unui program scris în limbajul C# este descrisă în figura următoare:

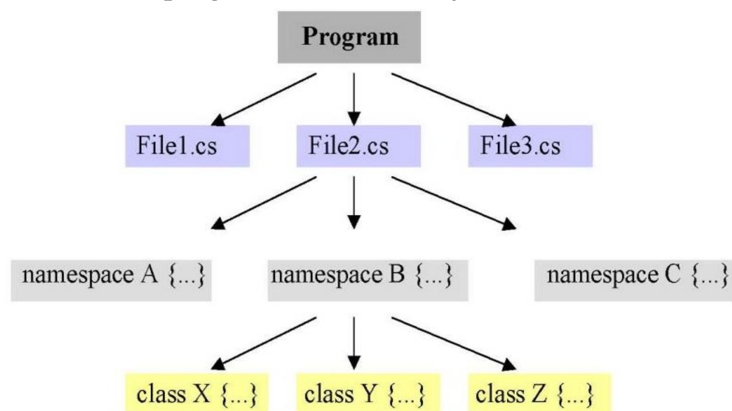


Figura 2. Structura unui program scris în limbajul C#

Următorul program C# afişează pe ecran mesajul "Hello World !":

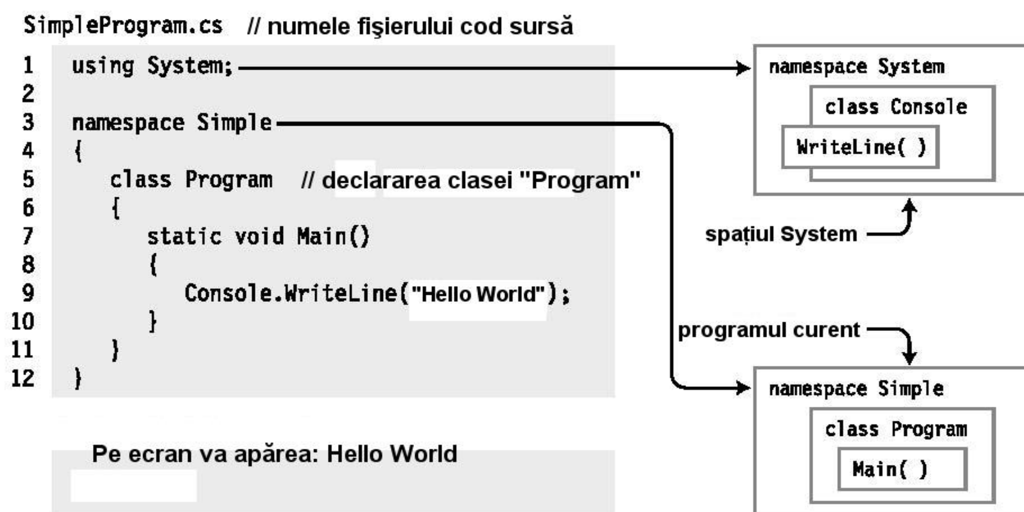


Figura 3. Exemplu de program C#

Un program C# este format din una sau mai multe clase, grupate într-un **spaţiu de nume (namespace)**. Un **spaţiu de nume** este o colecţie de clase care au asociat un nume. Acesta poate cuprinde mai multe clase cu nume diferite având funcţionalităţi înrudite. Două clase pot avea acelaşi nume cu condiţia ca ele să fie definite în nume de spaţii diferite. În cadrul aceluiaşi nume de spaţiu poate apărea definiţia unui alt nume de spaţiu, caz în care avem de-a face cu spaţii de nume imbricate. O **clasă** poate fi identificată prin numele complet (nume precedat de numele spaţiului sau spaţiilor de nume din care face parte clasa respectivă, cu separatorul punct). În exemplul nostru, Simple.Program este numele cu specificaţie completă a clasei Program.

Codul şi datele scrise în limbajul C# trebuie incluse într-o **clasă**. Nu se pot defini variabile şi nu se pot scrie declaraţii înafara claselor. O **clasă** este formată din date şi metode (funcţii). Toate clasele derivă din clasa de bază denumită ***object***. Apelarea unei metode în cadrul clasei în care a fost definită aceasta presupune specificarea numelui metodei. Apelul unei metode definite în interiorul unei clase poate fi invocată şi din interiorul altei clase, caz în care este necesară specificarea clasei şi apoi a metodei separate prin punct. Dacă în plus, clasa aparţine unui spaţiu de

nume neinclus în fişierul curent, atunci este necesară precizarea tuturor componentelor numelui: *spaţiu.clasă.metodă* sau *spaţiu.spaţiu.clasă.metodă*, etc.

În programul nostru de mai sus se află două spaţii de nume: unul definit (**Simple**) şi unul extern inclus prin directiva **using System;**. **Console.WriteLine** reprezintă apelul metodei **WriteLine** definită în clasa **Console**. Cum în spaţiul de nume curent este definită doar clasa **Program**, deducem că definiţia clasei **Console** trebuie să se găsească în spaţiul **System**. Altfel spus, denumirea completă ar fi **System.Console.WriteLine**, ceea ce însă, printre altele, ar fi incomod de redactat dacă apare adesea într-un program.

În cazul în care mai mulţi programatori lucrează la realizarea unei aplicaţii complexe, există posibilitatea de a segmenta aplicaţia în mai multe fişiere numite **assemblies**. Într-un **assembly** se pot implementa mai multe spaţii de nume, iar parţi ale unui aceeaşi spaţiu de nume se pot regăsi în mai multe **assembly-uri**. Pentru o aplicaţie consolă, ca şi pentru o aplicaţie Windows de altfel, este obligatoriu ca una (şi numai una) dintre clasele aplicaţiei să conţină un **„punct de intrare" (entry point),** şi anume **metoda (funcţia) Main**.

Să comentăm programul de mai sus:

- **linia 1:** spune compilatorului că acest program utilizează tipuri de date şi clase incluse în spaţiul de nume **System.** În cazul nostru se va folosi clasa **Console.**
- **linia 3**: se declară un nou spaţiu de nume, numit **Simple.** Noul spaţiu de nume începe la acolada deschisă din linia 4 şi extinde până la acolada închisă din linia 12. Orice tip declarat în această secţiune este membru al spaţiului de nume **Simple**.
- **linia 5:** orice program C# este alcătuit din una sau mai multe clase. În această linie este declarat un nou tip de clasă, denumită **Program.** Orice **membrii** declaraţi între acoladele care încep în linia 6 şi se termină în linia 11 sunt membrii care alcătuiesc această clasă.
- **linia 7:** în această linie este declarată **metoda (funcţia) Main** ca membru al clasei **Program**. În acest program, metoda (funcţia) **Main** este doar un membru al clasei **Program**. **Main** este o funcţie specială utilizată de compilator ca **„punctul de intrare"** în program.
- **linia 9:** Conţine doar o singură declaraţie simplă; această linie constituie „corpul" **funcţiei Main**. Această declaraţie foloseşte clasa numită **Console**, amintită mai sus, care aparţine spaţiului de nume **System** şi este folosită pentru **operaţiile de intrare/ieşire**. Aici se apelează metoda **WriteLine** din această clasă, pentru afişarea mesajului dorit pe ecran. Fără folosirea directivei din linia 1 - **using System** – compilatorul nu are de unde să ştie unde se găseşte clasa **Console.**

1.3 Metodele unui clase

O **metodă** este un bloc de cod format dintr-o serie de instrucţiuni care conduc la realizarea unei acţiuni. O metodă este asemănătoare cu **funcţiile, procedurile** sau **subrutinele** din limbajul C, C++ sau Pascal cu anumite excepţii. O **metodă** este o **funcţie** conţinută înăuntrul unei clase.

Metoda Main()

Orice aplicaţie scrisă în limbajul C# trebuie să conţină o metodă numită **Main(),** care reprezintă **punctul de intrare în aplicaţie**. În acest exemplu metoda Main() nu preia nici un

argument din linia de comandă şi nu returnează explicit un indicator de stare a terminării programului. Toate liniile de cod care formează corpul metodei Main() vor fi executate automat atunci când programul este rulat.

Clasa Console

Clasa **Console** aparţine de spaţiul de nume **System**. Cele două metode (funcţii de afişare pe ecran) **Write** şi **WriteLine** afişează pe consolă (ecran) argumentele din paranteză. În cazul metodei **WriteLine**, după afişare se trece la linie nouă. Următoarea linie de cod este un exemplu al utilizării metodei Write:

```
Console.Write("Acesta este un text.");
                         ↓
                   Output string
```

Codul scris mai sus produce următoarea ieşire pe ecran: `Acesta este un text.`

Un alt exemplu este următorul cod, care afişează pe ecran trei şiruri de caractere:

```
System.Console.Write ("This is text1.");
System.Console.Write ("This is text2.");
System.Console.Write ("This is text3.");
```

Acest cod produce următoarea ieşire:

```
This is text1. This is text2. This is text3.
```

Sintaxa metodei **Write** şi **WriteLine** este următoarea:

```
Console.WriteLine(FormatString, SubVal0, SubVal1, SubVal2, ... );
```

Dacă există mai mulţi parametrii, aceştia sunt separaţi prin virgulă. Primul parametru trebuie să fie întotdeauna un şir de caractere şi este denumit **şir de formatare (format string)**. Şirul de formatare poate conţine **marcatori de substituţie (substitution markers).** Un marcator de substituţie marchează poziţia în formatul şirului unde o valoare se substituie atunci când şirul este afişat pe ecran. Un marcator este un număr natural cuprins între acolade, şi care reprezintă poziţia numerică a valorii care urmează a fi substituită şi care începe de la 0. Parametrii care urmează după şirul de formatare se numesc **valori de substituţie (substitution values).**

Următorul cod conţine doi marcatori de substituţie, numărul 0 şi 1, şi două valori de care urmează a fi substituite, 3 şi 6.

```
                                          Substitution markers
                                               ↓      ↓
  Console.WriteLine("Doua exemple de intregi sunt {0} si {1}.", 3, 6);
                        ↑                                        ↑
                 Format string                        Substitution values
```

Acest cod produce pe ecran: `Doua exemple de intregi sunt 3 si 6.`

Un marcator nu trebuie să refere o valoare de la o poziţie mai lungă decât lista valorilor ce trebuie substituite, în caz contrar se produce o eroare. Următorul exemplu reflectă această situaţie:

```
                           Poziţia 0  Poziţia 1
                               ↓       ↓
Console.WriteLine("Doi intregi: {0} si {2}.", 3, 6); // Eroare!
                                       ↑
                               Nu există poziţia 2.
```

1.4 Comentarii

Comentariile sunt bucăţi de text care sunt excluse de la compilarea programului şi care servesc la introducerea în program a unor explicaţii referitoare la părţi ale acestuia.

- **comentariu pe un rând** prin folosirea // Tot ce urmează după caracterele // sunt considerate, din acel loc, până la sfârşitul rândului drept comentariu:

```
// Acesta este un comentariu pe un singur rând
```

- **comentariu pe mai multe rânduri** prin folosirea /* şi */ Orice text cuprins între simbolurile menţionate mai sus se consideră a fi comentariu. Simbolurile /* reprezintă începutul comentariului, iar */ sfârşitul respectivului comentariu:

```
/* Acesta este un
comentariu care se
întinde pe mai multe rânduri */
```

1.5 Declaraţii

O declaraţie simplă este o instrucţiune în cod sursă care descrie un tip de date sau care îi spune programului să execute o acţiune. O instrucţiune simplă se termină cu caracterul punct şi virgulă „;".

Următorul exemplu de cod cuprinde două instrucţiuni simple. Prima instrucţiune defineşte variabila cu numele „var1" şi o iniţializează cu valoarea 5. A doua instrucţiune afişează valoarea variabilei „var1" pe ecran:

```
int var1 = 5;
System.Console.WriteLine("Valoarea lui var1 este {0}", var1);
```

1.6 Blocuri de instrucţiuni

Un bloc de instrucţiuni cuprinde 0 sau mai multe instrucţiuni cuprinse între acolade {}.
Exemplu:

```
{
int var1 = 5;
System.Console.WriteLine("Valoarea lui var1 este {0}", var1);
}
```

Un bloc este folosit de obicei acolo unde mai multe instrucţiuni definesc o acţiune. Un bloc de instrucţiuni **nu** se termină cu punct şi virgulă „;".

```
  Terminating ;
             ↓                                      Terminating ;
{int var2 = 5;                                                  ↓
System.Console.WriteLine("The value of var1 is {0}", var1);
}
↑ No terminating ;
```

1.7 Identificatori

Prin nume dat unei variabile, clase, metode etc. înţelegem o succesiune de caractere care îndeplineşte următoarele reguli:

- numele trebuie să înceapă cu o literă sau cu unul dintre caracterele "_" şi "@";
- primul caracter poate fi urmat numai de litere, cifre sau un caracter de subliniere;
- numele care reprezintă cuvinte cheie nu pot fi folosite în alt scop decât acela pentru care au fost definite;
- cuvintele cheie pot fi folosite în alt scop numai dacă sunt precedate de @;

- două nume sunt distincte dacă diferă prin cel puţin un caracter (fie el şi literă mică ce diferă de aceeaşi literă majusculă);

Convenţii pentru nume:

- în cazul numelor claselor, metodelor, a proprietăţilor, enumerărilor, interfeţelor, spaţiilor de nume, fiecare cuvânt care compune numele începe cu majusculă;
- în cazul numelor variabilelor dacă numele este compus din mai multe cuvinte, primul începe cu minusculă, celelalte cu majusculă.

Simbolurile lexicale reprezentând constante, regulile de formare a expresiilor, separatorii de liste, delimitatorii de instrucţiuni, de blocuri de instrucţiuni, de şiruri de caractere etc. sunt în mare aceiaşi ca şi în cazul limbajului C++.

Secvenţe escape

Secvenţele escape permit specificarea caracterelor care nu au reprezentare grafică şi reprezentarea unor caractere speciale precum backslash, apostrof, etc.

Secvenţele escape predefinite în C# sunt:

– \b : Backspace (BS)
– \t : Tab orizontal (HT)
– \n : Linie noua (LF)
– \f : Pagina noua (FF)
– \r : Inceput de rând (CR)
– \" : Ghilimele
– \’ : Apostrof
– \\ : Backslash

Literali “copie la indigo”

Un astfel de literal începe cu caracterul @ urmat de un şir de caractere cuprins între ghilimele. Specificul acestui literal constă în faptul că acesta va putea fi reprezentat exact aşa cum este scris, inclusiv dacă se întinde pe mai multe rânduri. Pentru a afişa “ va trebui folosit ”” (ghilimele duble).

Exemplu:

```
Console.Write (@”Ne
                    place
                        ““informatica”” mult”);
```

Pe ecran va apărea:

```
Ne
  place
       ”informatica” mult
```

1.8 Cuvinte cheie

Acestea sunt cuvinte speciale care au o semnificaţie bine definită şi care nu pot fi folosite decât în scopul pentru care au fost create.

abstract	const	extern	int	out	short	typeof
as	continue	false	interface	override	sizeof	uint
base	decimal	finally	internal	params	stackalloc	ulong
bool	default	fixed	is	private	static	unchecked

break	delegate	float	lock	protected	string	unsafe
byte	do	for	long	public	struct	ushort
case	double	foreach	namespace	readonly	switch	using
catch	else	goto	new	ref	this	virtual
char	enum	if	null	return	throw	void
checked	event	implicit	object	sbyte	true	volatile
class	explicit	in	operator	sealed	try	while

Figura 4. Cuvintele cheie din C#

1.9 Tipuri de date

În limbajul de programare C# există două tipuri de date: **tipuri valoare şi tipuri referinţă**. **Tipurile valoare** includ tipurile simple (char, int, float, etc), tipurile enumerare şi structură acestea conţinând direct datele referite şi sunt alocate pe stivă sau inline într-o struct. **Tipurile referinţă** includ tipurile clasă, interfaţă, delegat şi tablou, toate având proprietatea că variabilele de acest tip stochează referinţe către obiecte. Toate tipurile de date sunt derivate (direct sau nu) din tipul System.Object.

Tipuri predefinite

Limbajul C# conţine un set de 15 tipuri predefinite, pentru care nu este necesară includerea vreunui spaţiu de nume via directiva using: string, object, tipurile întregi cu semn şi fără semn, tipuri numerice în virgulă mobilă, tipurile bool şi decimal.

Tipul *string* este folosit pentru manipularea şirurilor de caractere codificate Unicode; conţinutul obiectelor de tip string nu se poate modifica.

Clasa *object* este rădăcina ierarhiei de clase din .NET, la care orice tip (inclusiv un tip valoare) poate fi convertit.

Tipul *bool* este folosit pentru a reprezenta valorile logice true şi false.

Tipul *char* este folosit pentru a reprezenta caractere Unicode, reprezentate pe 16 biţi.

Tipul *decimal* este folosit pentru calcule în care erorile determinate de reprezentarea în virgulă mobilă sunt inacceptabile, de exemplu în calcule monetare, el punând la dispoziţie 28 de cifre zecimale semnificative. Pentru a semnala că un literal în virgulă mobilă este de tip decimal, se va pune la sfârşitul acestuia litera m sau M.

Exemplu: decimal dcm=123.456m;

Pentru a semnala că un literal în virgulă mobilă este de tip float, se va pune la sfârşitul acestuia litera f sau F, altfel se va subînţelege că este de tip double şi ar putea apărea erori.

Exemplu: float flt=345.678F;

Denumire	Explicaţie	Valori reprezentate
sbyte	întreg cu semn pe 8 biţi	-128 ÷ 127
byte	întreg fără semn pe 8 biţi	0 ÷ 255
short	întreg cu semn pe 16 biţi	-32768 ÷32767
ushort	întreg fără semn pe 16 biţi	0 ÷ 65535
int	întreg cu semn pe 32 biţi	-2.147.483.648 ÷ 2.147.483.647
uint	întreg fără semn pe 32 biţi	0 ÷ 4.294.967.295
long	întreg cu semn pe 64 biţi	-9.223.372.036.854.775.808 ÷ 9.223.372.036.854.775.807

ulong	întreg fără semn pe 64 biţi	0 ÷ 18.446.744.073.709.551.615
float	real cu precizie simplă	$1.5x10^{-45} \div 3.4x10^{38}$
double	real cu precizie dublă	$5x10^{-324} \div 1.7x10^{308}$
bool	logic	true, false
char	caracter Unicode	U+0000 ÷ U+ffff
decimal	128 de biţi, valoare reală cu precizie de 28 de zecimale semnificative	$\pm1.0x10^{-28} \div \pm7.9x10^{28}$
object	clasa de bază din care derivă toate tipurile	System.Object
string	secvenţă de caractere Unicode	System.String
dynamic	tip folosit pentru assembly-urile din limbaje dinamice	

Figura 5. Tipuri predefinite în C#

Fiecare tip de date din C# îşi are corespondentul în tipul .Net (tipurile din spaţiul de nume System).

Tipuri definite de utilizator

Tipurile care pot fi definite de utilizatori sunt de 6 feluri. O parte dintre acestea vor fi discutate în paginile următoare.

- class;
- struct;
- array;
- enum;
- delegate;
- interface.

1.10 Variabile

Un limbaj de programare trebuie să permită programului să stocheze şi să manipuleze date. În timpul execuţiei unui program, datele sunt temporal stocate în memorie. O variabilă este un nume dat unei locaţii de memorie folosită de un tip particular de date în timpul execuţiei programului. Astfel, fiecărei variabile i se asociază un tip de dată şi o valoare. C# prevede patru categorii de variabile, fiecare dintre acestea vor fi discutate în detaliu:

- Variabile locale;
- Câmpuri;
- Parametrii;
- Elemente de tablou.

Declararea variabilelor. În C# variabilele sunt declarate astfel:

```
<nume_tip_dată>  <nume_variabilă>;
```

Exemplu:

Tip Nume variabilă
↓ ↓

```
int var2;
```

Codul de program de mai sus rezervă o arie de 4 bytes(octeţi) în memoria RAM, pentru stocarea valorii unei valori de tip întreg, care va fi referită în program cu ajutorul identificatorului „var2”. Variabilele pot fi iniţializate la declarare sau mai târziu:

```
bool isReady = true;
float percentage = 87.88, average = 43.9;
char digit = '7';
```

După declararea tipului unei variabile, aceasta poate fi iniţializată cu o valoare. Iniţializarea unei variabile se face cu ajutorul semnului egal („=") plasat între numele variabilei şi valoarea acesteia, aşa cum se arată în continuare:

```
                                   iniţializare
                                        ↓
                          int var2 = 17;
```

Variabilele locale neiniţializate au valori nedefinite şi prin urmare nu pot fi folosite până când nu au atribuită o valoare. Încercarea de a utiliza o variabilă neiniţializată va produce o eroare la compilare.

Declararea mai multor variabile de acelaşi tip. Se pot declara mai multe variabile într-o declaraţie. Toate variabilele dintr-o declaraţie de variabile, trebuie să fie de acelaşi tip. Numele variabilelor trebuie separate prin virgulă iniţializarea acestora se poate face cu ajutorul semnului „=". În următorul cod de program se declară se declară, mai multe variabile de tip întreg şi respectiv real:

```
// Declaraţii de variabile - cu iniţializare şi fără
int var3 = 7, var4, var5 = 3;
double var6, var7 = 6.52;
Tip        Tip diferit
 ↓           ↓
int var8, float var9; //Error!
```

Ultima declaraţie este invalidă deoarece sunt declarate două variabile de tipuri diferite, în aceeaşi instrucţiune.

Utilizarea valorii unei variabile. Numele unei variabile reprezintă valoarea stocată de variabilă. Se poate utiliza valoarea prin folosirea numelui variabilei. De exemplu, valoarea variabilei var este luată din memorie şi plasată în poziţia numelui variabilei, astfel:

```
Console.WriteLine("{0}", var);
```

În C# (la fel ca şi în celelalte limbaje de programare moderne), variabilele trebuie declarate înaintea folosirii acestora. Următorul cod de program va da eroare la compilare, deoarece variabila „număr" nu este iniţializată:

```
static void Main()
{
 int număr;
 //număr = 18;
 Console.WriteLine(număr); // error
}
```

În cazul în care se vor şterge cele două // care preced comentariul din linia 4, codul poate fi compilat fără eroare.

O variabilă de un *tip valoare* conţine efectiv o valoare. Când se declară o variabilă de un tip valoare, compilatorul alocă în stivă numărul de octeţi corespunzători tipului, iar programatorul lucrează direct cu această zonă de memorie.

O variabilă de *tip referinţă* este o referinţă care, atunci când nu este *null*, referă un obiect de tipul specificat, alocat în memoria *heap*.

1.11 Constante. Constantele sunt variabile a căror valoare, odată definită, nu poate fi schimbată de către program.

Constantele sunt declarate astfel: `const double PI = 3.142;`

CAPITOLUL II. EXPRESII ŞI OPERATORI

2.1 Expresii

Expresiile reprezintă înşiruiri de operatori şi operanzi. Operanzii pot fi: literali (constante), variabile, apeluri de metode, elemente de tablou, etc.

Operatori

2.2 Operatori aritmetici

Operand	Descriere
+	Adunare
-	Scădere
*	Înmulţire
/	Împărţire
%	Restul sau modulo
++	Adunare cu 1
--	Scădere cu 1

Notaţii prefixate sau postfixate

Se dă următoarea linie de cod: `int num1 = 10;`

Următoarele instrucţiuni sunt echivalente:

```
num1++;
num1 = num1 + 1;
num1 += 1;
```

Ambii operatori ++ şi – pot fi utilizaţi în formă prefixată sau postfixată.

În forma *prefixată*,

```
num1 = 3;
num2 = ++num1; // num1 = 4, num2 = 4
```

compilatorul va incrementa prima dată variabila num1 cu 1 şi apoi va atribuii această valoare variabilei `num2`.

În forma *postfixată*,

```
num2 = num1++; // num1 = 4, num2 = 3
```

compilatorul prima dată va atribuii valoarea variabilei `num1` la variabila `num2` şi apoi incrementează variabila `num1` cu 1.

2.3 Operatori de atribuire

Operand	Descriere
=	Atribuire simplă
+=	Atribuire aditivă
-=	Atribuire substractivă
*=	Atribuire multiplicativă
/=	Atribuire cu împărţire

%=	Atribuire cu modulo

O atribuire compusă are forma: <var> op= <expr> şi este echivalentă cu: <var> = <var> op <expr>, iar op poate fi unul din operatorii: *, /, %, +, -, <<, >>, &, |, ^

Exemplu:

```
int i=0;
i += 15;  echivalent cu i = i + 15;
```

Exemplu:

```
using System;
class FormattingNumbers
{
    static void Main()
    {
        int val = 10;
        val += 10;
        Console.WriteLine("val +=10 este {0}", val);  //val = 20
        val -= 5;
        Console.WriteLine("val -=5 este {0}", val);   //val = 15
        val *= 10;
        Console.WriteLine("val *=10 este {0}", val);  //val = 150
        val /= 3;
        Console.WriteLine("val /=3 este {0}", val);   //val = 50
        val %= 8;
        Console.WriteLine("val %=8 este {0}", val);   //val = 2
        Console.ReadLine();
    }
}
```

Citirea datelor de la tastatură şi scrierea datelor la consolă

Citirea unui şir de caractere de la tastatură se face cu metodele **Console.Read()** sau **Console.ReadLine()**. Afişarea datelor pe ecran se face cu metodele **Console.Write()** sau **Console.WriteLine()**. Diferenţa dintre cele două constă în trecerea la linie nouă în cazul metodelor care se termină cu „Line”.

Exemplu:

```
using System;
namespace ConsoleApplication
{
    class Program
    {
        public static void Main()
        {
            float a, b;
            string sir;
            Console.Write("Dati valoarea pentru variabila a=");
            sir = Console.ReadLine(); a = float.Parse(sir);
            Console.Write("Dati valoarea pentru variabila b=");
            sir = Console.ReadLine(); b = float.Parse(sir);
            Console.Write("Suma dintre a={0} şi b={1} este {2}", a, b, a+b);
            Console.ReadLine();
        }
    }
```

```
}
```
Exemplul de mai sus se poate scrie şi astfel:
```
using System;
namespace ConsoleApplication
{
 class Program
 {
  public static void Main()
   {
    float a, b;
    Console.Write("Dati valoarea pentru variabila a=");
    a = float.Parse(Console.ReadLine());
    Console.Write("Dati valoarea pentru variabila b=");
    b = float.Parse(Console.ReadLine());
    Console.Write("Suma dintre a={0} şi b={1} este {2}", a, b, a+b);
   Console.ReadLine();
   }
 }
}
```

Pentru tipurile numerice, metoda **ToString** produce o reprezentare şir de caractere a unui număr. Ea poate fi folosită şi cu câţiva parametrii, care permit diverse formatări ale afişării valorii numerice ca şi şir de caractere.

În C# există predefinite o mulţime de modalităţi pentru formatarea afişării valorilor numerice. Lista completă a acestora se poate găsi la adresele de Internet:
http://msdn.microsoft.com/en-us/library/dwhawy9k.aspx
http://msdn.microsoft.com/en-us/library/0c899ak8.aspx

Practic este vorba de nişte specificatori de format, reprezentaţi de obicei de o literă, însă pot fi şi cifre sau alte caractere (# % . ,).

De exemplu, specificatorul de format c (sau C) va face ca valoarea numerică afişată să conţină şi un simbol monetar.
```
decimal value = 123.456m;
Console.WriteLine(value.ToString("C2"));
```
Va afişa: `$123.46`

Complementara metodei **ToString** este metoda **Parse**. Metoda Parse se foloseşte astfel:
```
variabilă = tipul_variabilei.Parse(input_string);
```
unde `tipul_variabilei` poate fi oricare tip de date predefinit în C#.

Exemplu:
```
int k = int.Parse("12345");
double dd = double.Parse(" -1,234.5678 ");
Console.WriteLine("k={0} dd={1}", k, dd);
```

Exemplu:
```
using System;
class Program
{
    static void Main()
    {
        int a = 55000;
```

```
        int b = 676;
        double c = 555.4;
        double d = 50870.1155;
        string a1 = a.ToString();
        string a2 = a.ToString();
        Console.WriteLine(a1 + " " + a2);    // 55000 55000
        string b1 = b.ToString();
        string b2 = b.ToString();
        Console.WriteLine(b1 + " " + b2);      // 676 676
        string c1 = c.ToString();
        string c2 = c.ToString();
        Console.WriteLine(c1 + " " + c2);      // 555.4 555.4
        string d1 = d.ToString();
        string d2 = d.ToString();
        Console.WriteLine(d1 + " " + d2);      // 50870.1155 50870.1155
        Console.ReadLine();
    }
}
```

2.4 Operatori relaţionali

Operand	Descriere
==	Testarea egalităţii
!=	Testarea inegalităţii
>	Strict mai mare
<	Strict mai mic
>=	Mai mare sau egal
<=	Mai mic sau egal

Exemplu:

```
using System;
class OperatoriRelationali
{
 static void Main()
 {
  int v1 = 5, v2 = 6;
  Console.WriteLine("{0}=={1} este {2}", v1, v2, v1==v2); //false
  Console.WriteLine("{0}!={1} este {2}", v1, v2, v1!=v2); //true
  Console.WriteLine("{0}>{1} este {2}", v1, v2, v1>v2);   //false
  Console.WriteLine("{0}<{1} este {2}", v1, v2, v1<v2);   //true
  Console.WriteLine("{0}<={1} este {2}", v1, v2, v1<=v2); //true
  Console.WriteLine("{0}>={1} este {2}", v1, v2, v1>=v2); //false
  Console.ReadLine();
 }
}
```

2.5 Operatori logici. Operatori logici la nivel de biţi

Operand	Descriere
&	AND între biţi
|	OR între biţi
^	XOR între biţi

~	NOT pe biţi
!	NOT logic
&&	AND logic
||	OR logic

Operatori logici „scurtcircuitaţi”: &&, ||

Aceşti operatori se numesc „scurtcircuitaţi” deoarece în cazul lui &&, dacă primul operand este fals, rezultatul evaluării expresiei este automat fals, fără a mai verifica valoarea celui de-al doilea operand. În cazul lui ||, dacă primul operand este adevărat, rezultatul evaluării expresiei este automat adevărat, fără a mai verifica valoarea celui de-al doilea operand. Neevaluând al doilea operand se câştigă timp la execuţie.

Exemplu:
```
using System;
class OperatoriLogici
{
 static void Main()
  {
   int i = 6, j = 12;
   bool firstVar = i > 3 && j < 10;
   Console.WriteLine("{0}>3 && {1}<10 este {2}", i, j, firstVar);
   // firstVar va avea valoarea false
   bool secondVar = i > 3 || j < 10;
   Console.WriteLine("{0}>3 || {1}<10 este {0}", i, j, secondVar);
   // secondVar va avea valoarea true
   Console.ReadLine();
  }
}
```

2.6 Operatorul condiţional ?:

Acesta este un operator care necesită trei operanzi (operator terţiar). Forma generală a acestuia este: `condiţie ? expresie1 : expresie2`

Rolul acestui operator este de a evalua o `condiţie` şi în funcţie de valoarea de adevăr a acesteia să execute (dacă este adevărată) `expresie1` sau `expresie2` (dacă este falsă).

Exemplu:
```
int i, x=3, y=4;
i = x < y ? 10 : 20 ;
```
Efect: Dacă x<y atunci i va lua valoarea 10, altfel va lua valoarea 20.

Precedenţa (prioritatea) operatorilor

În cazurile când avem expresii cu mai mulţi operatori este util să cunoaştem ordinea în care aceştia vor fi evaluaţi, mai ales când nu sunt folosite paranteze. În tabelul de mai jos sunt operatorii, de la cea mai mare prioritate spre cea mai mică şi asociativitatea când sunt operatori cu aceeaşi prioritate.

Operator	Asociativitate
`() [] .`	stânga spre dreapta

++(postfix) --(postfix) checked new sizeof unchecked	
++(prefix) --(prefix) +(unar) -(unar) (cast) ! ~	dreapta spre stânga
* / %	stânga spre dreapta
+ -	stânga spre dreapta
<< >>	stânga spre dreapta
< < > >= is	stânga spre dreapta
== !=	stânga spre dreapta
&	stânga spre dreapta
^	stânga spre dreapta
|	stânga spre dreapta
&&	stânga spre dreapta
||	stânga spre dreapta
?:	dreapta spre stânga
= += -= *= /= %= &= ^= |= <= >>=	dreapta spre stânga

CAPITOLUL III. INSTRUCŢIUNI

Instrucţiuni condiţionale

3.1 Instrucţiunea `if-else`

Instrucţiunea `if` execută o instrucţiune în funcţie de valoarea de adevăr a unei expresii logice. Structura instrucţiunii `if…else` este următoarea:

```
 if (expresie_booleană)
      instrucţiune sau bloc de instrucţiuni 1;
[else
      instrucţiune sau bloc de instrucţiuni 2;]
```

Clauza `else` este opţională.

Exemplu:

```
int a=10;
if (a<0)
  Console.Write (”Negativ”);
else
  Console.Write (”Pozitiv”);
```

3.2 Instrucţiunea `switch`

În cazul unei instrucţiuni `switch`, expresia care se evaluează trebuie să fie de tipul (sau convertibilă implicit la tipul) `sbyte, byte, short, ushort, long, ulong, char, string` sau o enumerare bazată pe unul dintre aceste tipuri. Dacă valoarea expresiei se regăseşte printre valorile specificate la clauzele `case`, atunci instrucţiunea corespunzătoare va fi executată; dacă nu, atunci instrucţiunea de la clauza `default` va fi executată (dacă ea există).

```
switch (expresie)
case eticheta1: instrucţiune sau bloc de instrucţiuni;
    instrucţiunea break sau goto;
.
.
case etichetaN: instrucţiune sau bloc de instrucţiuni;
    instrucţiunea break sau goto;
[default]
instrucţiune sau bloc de instrucţiuni;
            instrucţiunea break sau goto;
```

O etichetă reprezintă o expresie constantă.

În faza de proiectare a limbajului C#, Microsoft a decis să împiedice trecerea automată la următoarea ramură case în lipsa instrucţiunii break (o facilitate uzuală în C şi C++); din acest motiv, lipsa instrucţiunii `break` pe o ramură `case` va genera eroare la compilare (exceptând cazul când ramura `case` nu conţine instrucţiuni). Trecerea de la o ramură `case` la alta poate fi însă simulată cu instrucţiunea **`goto`**: ...

O instrucţiune poate să şi lipsească şi în acest caz se va executa instrucţiunea de la `case`–ul următor, sau de la `default`. Secţiunea `default` poate să lipsească. Dacă o instrucţiune este nevidă,

atunci obligatoriu va avea la sfârşit o instrucţiune `break` sau `goto case expresieConstanta` sau `goto default`.

Spre deosebire de C şi C++, e interzis să se folosească trecerea automată de la o etichetă la alta, continuarea se face folosind explicit instrucţiunea `goto`.

Exemplu:

```
using System;
class TestSwitch
{
 static void Main()
  {
   int nota=8;
   switch (nota) {
      case 4: Console.Write ("Nepromovat"); break;
      case 8: Console.Write ("Mediu"); break;
      case 10: Console.Write ("Foarte bine"); break;
   }
   Console.ReadLine();
  }
}
```

Instrucţiuni de ciclare

3.3 Instrucţiunea **`for`**

Formatul instrucţiunii este:

```
for(expresie_de_iniţializare; condiţie; incrementare/decrementare)
      instrucţiune sau bloc de instrucţiuni;
```

Exemplu:

```
for(int i=1; i<=10; i++)
 {
  Console.WriteLine("In acest ciclu valoarea lui i este {0}.", i);
 }
```

Toate cele trei argumente ale instrucţiunii `for` sunt opţionale.

Exemple:

```
for(; ;)
for( ; i<10; i++)
for(int i=3; ; i--)
for( ; i>5; )
```

Pot exista forme al instrucţiunii `for` care folosesc mai multe contoare, separate între ele prin virgulă, după cum se poate vedea în exemplul de mai jos:

```
for(i=0, j=10; i<j; i++, j--)
```

Dacă se declară o variabilă în `expresia_de_iniţializare`, atunci ciclul ei de viaţă este doar în corpul instrucţiunii `for`.

Exemplu:

```
using System;
namespace exemplu_for
{
  class ExempluFor
    {
      static void Main()
```

```
        {
          for (int i = 1; i <= 10; i++)
           Console.WriteLine("Valoarea lui i este: {0}.", i);
           Console.ReadLine();
           i++; // eroare !!
        }
    }
}
```

Se pot utiliza instrucţiunile `break` şi `continue` în orice instrucţiune de ciclare. Acestea schimbă execuţia normală a ciclului. Instrucţiunea `break` termină ciclul şi transferă execuţia în afara ciclului.

Exemplu:

```
for(int i=1; i<=10; i++)
{
 if(i>5)
 {
  break;
 }
 Console.WriteLine("Valoarea lui i in acest ciclu este:{0}.", i);
}
```

Dacă după instrucţiunea `break` mai există cel puţin o instrucţiune, atunci compilatorul va genera o avertizare.

Exemplu:

```
for(int i=3; i<10; i++)
      {
      break; //warning, Console.WriteLine (i); is unreachable code
      Console.WriteLine(i);
      }
```

Instrucţiunea `continue` ignoră partea de instrucţiuni rămasă din iteraţia curentă şi trece la următoarea iteraţie.

Exemplu:

```
for(int i=1; i<=4; i++)
{
 if(i==2)
 {  continue;  }
  Console.WriteLine("Valoarea lui i este:{0}.", i);
}
```

Rezultatul:

```
Valoarea lui i in acest ciclu este:1.
Valoarea lui i in acest ciclu este:3.
Valoarea lui i in acest ciclu este:4.
```

3.4 Instrucţiunea `do-while`

Structura acestei instrucţiuni este:

```
do
      instrucţiune sau bloc de instrucţiuni
while(condiţie);
```

Se vor executa instrucţiunile de după `do`, după care se verifică condiţia. Instrucţiunea de ciclare continuă atâta timp cât condiţia este adevărată (true). Următorul program afişează întregii de la 1 la 10 pe ecran:

```
using System;
namespace exemplu_Do {
    class DoExample {
        static void Main() {
            int i = 1;
            do {
                Console.WriteLine("Valoarea lui i este {0}.", i);
                i++;
            } while (i <= 10);
            Console.ReadLine();
        }
    }
}
```

În cazul instrucţiunii `do-while`, instrucţiunile din corpul acesteia se execută cel puţin odată.

3.5 Instrucţiunea `while`

Instrucţiunea `while` este similară cu instrucţiunea `do-while`, cu excepţia că verificarea condiţiei se face înaintea execuţiei codului din corpul instrucţiunii. Forma generală a instrucţiunii este:

```
while(condiţie)
      instrucţiune sau bloc de instrucţiuni
```

Exemplu:

```
using System;
namespace exemplu_While {
    class WhileExample {
        static void Main() {
            int i = 1;
            while (i <= 10)
            {
                Console.WriteLine("Valoaea lui i este {0}.", i);
                i++;
            }
            Console.ReadLine();
        }
    }
}
```

3.6 Instrucţiunea `foreach`

Instrucţiunea `foreach` este un alt tip de instrucţiune de ciclare, care enumeră elementele dintr-o colecţie sau matrice (array), executând o instrucţiune pentru fiecare element. Elementul care se extrage este de tip read-only, neputând fi transmis ca parametru şi nici aplicat un operator care sa-i schimbe valoarea.

Structura de bază a instrucţiunii `foreach` este:

```
foreach(<tip_colecţie> <nume_variabilă> in <array sau colecţie>)
  <instrucţiune sau bloc de instrucţiuni>
```

Exemplu:

```
int[] t = {1, 2, 3};
foreach (int x in t)
 {
 Console.WriteLine(x);
 }
```

Exemplu:

```
string[] fructe={"Mar", "Ciresa", "Portocala"};
foreach (string fruct in fructe)
 {
 Console.Write("{0} ", fruct);
 }
```

Exemplu:

```
static void Main()
  {
  // declararea şi iniţializarea unui şir de întregi
  int [] integers = {3, 7, 2, 14, 65};
  // iterarea elementelor din şir şi tipărirea lor pe ecran
  foreach(int i in integers)
  {
   Console.WriteLine(i);
  }
}
```

În instrucţiunea foreach(int i in integers), se specifică tipul elementelor din colecţie (int în acest caz). Se declară variabila i care va conţine, la fiecare iteraţie, valoarea fiecărui element al colecţiei.

Observaţii:

- Variabila utilizată pentru a memora, la fiecare iteraţie, valoarea unui element al colecţiei, (i în cazul de mai sus) este de tip read only, deci nu se pot schimba valorile elementelor colecţiei prin intermediul ei. Acest fapt înseamnă că instrucţiunea `foreach` nu permite schimbarea valori elementelor colecţiei sau tabloului, ci doar parcurgerea acesteia/acestuia.
- Colecţia poate să fie orice instanţă a unei clase care implementează interfaţa System.Collections.IEnumerable.
- clasa string este, de asemenea, o colecţie de caractere.

```
static void Main()
 {
 string name = "Alexandru Cel Mare";
 foreach(char ch in name)
  {
  Console.WriteLine(ch);
  }
 }
```

Instrucţiuni de salt

3.7 Instrucţiunea **break**

Instrucţiunea `break` permite ieşirea forţată dintr-un ciclu de tip `switch`, `while`, `do-while`, `for` sau `foreach`.

Exemplu de folosire a instrucţiunii `break` într-un ciclu `for`:

```
using System;
class InstrBreak {
    static void Main() {
        for (int i = 1; i <= 100; i++)
        {
            if (i == 5) break;
            Console.WriteLine(i);
        }
        Console.ReadLine();
    }
}
```

Exemplu de folosire a instrucţiunii break într-un ciclu switch.

```
using System;
class Switch {
    static void Main() {
        Console.Write("Alegeti un numar (1, 2, or 3): ");
        string s = Console.ReadLine();
        int n = Int32.Parse(s);
        switch (n)
        {
            case 1:
                Console.WriteLine("Valoarea este {0}", 1);
                break;
            case 2:
                Console.WriteLine("Valoarea este {0}", 2);
                break;
            case 3:
                Console.WriteLine("Valoarea este {0}", 3);
                break;
            default:
                Console.WriteLine("Selectie gresita.");
                break;
        }
    }
}
```

Exemplu de folosire a instrucţiunii **`break`** într-un ciclu **`for`.**

```
using System;
class MainClass
{
    public static void Main()
    {
        // se va folosi break pentru a ieşi din ciclu
```

```
        for (int i = -10; i <= 10; i++)
        {
            if (i > 0) break; //ciclul se termină când i e pozitiv
            Console.Write(i + " ");
        }
        Console.WriteLine("Gata");
        Console.ReadLine();
    }
}
```

Exemplu de folosire a instrucţiunii `break` într-un ciclu `do-while`.

```
using System;
class MainClass
{
    public static void Main()
    {
        int i;
        i = -10;
        do
        {
            if (i > 0)
                break;
            Console.Write(i + " ");
            i++;
        } while (i <= 10);
        Console.WriteLine("Gata");
        Console.ReadLine();
    }
}
```

Exemplu de folosire a instrucţiunii `break` într-un ciclu `foreach`**.**

```
using System;
class MainClass
{
    public static void Main()
    {
        int sum = 0;
        int[] numere = new int[10];
        for (int i = 0; i < 10; i++)
            numere[i] = i;
        foreach (int x in numere)
        {
            Console.WriteLine("Valoarea este: " + x);
            sum += x;
            if (x == 4)
                break; // cand x=4 se iese din ciclu
        }
        Console.WriteLine("Suma primelor 5 elemente: " + sum);
    }
}
```

3.8 Instrucţiunea `continue`

Instrucţiunea `continue` ignoră partea rămasă din iteraţia curentă şi porneşte următoarea iteraţie, într-o instrucţiune de ciclare de tip `switch, while, do-while, for` sau `foreach`.

Exemplu de folosire a instrucţiunii continue, pentru afişarea numerelor pare dintre 0 şi 100.

```
using System;
class MainClass
{
    public static void Main()
    {
        // afiseaza numerele impare din intervalul 0, 100
        for (int i = 0; i <= 100; i++)
        {
            if ((i % 2) != 0)
                continue; // iterate
            Console.WriteLine(i);
        }
        Console.ReadLine();
    }
}
```

3.9 Instrucţiunea `goto`

Această instrucţiune permite saltul la o anumită instrucţiune. Are 3 forme:

```
goto eticheta;
goto case expresieconstanta;
goto default;
```

Cerinţa este ca eticheta la care se face saltul să fie definită în cadrul funcţiei curente şi saltul să nu se facă în interiorul unor blocuri de instrucţiuni, deoarece nu se poate reface întotdeauna contextul acelui bloc. Se recomandă evitarea utilizării intense a acestei instrucţiuni, în caz contrar se poate ajunge la fenomenul de ”spagetti code”. (a se vedea articolul clasic al lui EdsgerW. Dijkstra, ”Go To Statement Considered Harmful”:
http://www.acm.org/classics/oct95/).

Exemplu de folosire a instrucţiunii `goto`, în cazul instrucţiunii `switch`.

```
using System;
class SwitchGoto
{
    public static void Main()
    {
        for (int i = 1; i < 5; i++)
        {
            switch (i)
            {
                case 1:
                    Console.WriteLine("In case 1");
                    goto case 3;
                case 2:
                    Console.WriteLine("In case 2");
                    goto case 1;
```

```
                case 3:
                    Console.WriteLine("In case 3");
                    goto default;
                default:
                    Console.WriteLine("In default");
                    break;
            }
            Console.WriteLine();
        }
    }
}
```

Exemplu de folosire a instrucţiunii `goto`, în cazul instrucţiunii `for`.

```
using System;
class MainClass
{
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("i" + i);
            if (i == 3)
                goto stop;
        }
        stop:
        Console.WriteLine("Stopped!");
    }
}
```

Exemplu de folosire a instrucţiunii `goto`, în cazul instrucţiunii `while`.

```
using System;
class MainClass
{
    static void Main()
    {
        int a = 0;
        while (a < 10)
        {
            if (a == 5)
                goto cleanup;
            a++;
        }
    cleanup:
        Console.WriteLine(a);
        Console.ReadLine();
    }
}
```

Exemplu de folosire a instrucţiunii `goto`, în cazul instrucţiunii `if`.

```
using System;
class MainClass {
   public static void Main() {
        int total = 0;
        int counter = 0;
```

```
    myLabel:
        counter++;
        total += counter;
        if (counter < 5)
        {
            System.Console.WriteLine(counter);
            goto myLabel;
        }
        Console.ReadLine();
    }
}
```

3.10 Instrucţiunea `return`

Instrucţiunea `return` termină execuţia metodei în care aceasta apare, şi returnează controlul metodei apelante. Instrucţiunea `return` poate returna, opţional, o valoare. Dacă metoda care conţine instrucţiunea `return` este de tip `void`, atunci instrucţiunea `return` poate fi omisă.
Forma generală a instrucţiunii este:

```
return [expression];
```

unde `expression` este valoarea returnată de metodă. Nu este utilizată cu metode de tip `void`.

În exemplul următor, metoda `A()` returnează variabila `Area` ca valoare de tip `Double`:

```
using System;
class ReturnTest
{
    static double CalculateArea(int r)
    {
        double area;
        area = r * r * Math.PI;
        return area;
    }
    public static void Main()
    {
        int raza = 5;
        Console.WriteLine("Aria e {0:0.00}", CalculateArea(raza));
        Console.ReadLine();
    }
}
```

Indentarea programelor

Limbajul C# este independent de format, în sensul că nu are importanţă unde sunt poziţionate instrucţiunile unele în raport cu celelalte. De-a lungul timpului însă a fost dezvoltat şi adoptat la scară largă un sistem de indentare comun, care permite apoi citirea programelor mult mai uşor. Acest sistem presupune trecerea la un nou nivel de indentare după fiecare acoladă deschisă şi revenirea la nivelul de indentare anterior la închiderea acoladei. Există de asemenea unele instrucţiuni care necesită o indentare suplimentară. Se recomandă ca programatorii să adopte acest stil de scriere a programelor.

CAPITOLUL IV. TIPURI DEFINITE DE UTILIZATOR

4.1 Tipul enumerare

Tipul enumerare este un tip definit de utilizator. Tipul enumerare este un tip valoare, construit pentru a permite declararea constantelor înrudite, într-o manieră clară şi sigură din punct de vedere al tipului. Un exemplu este:

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

În această enumerare `Sat` este 0, `Sun` este 1,… Elementele enumerării pot fi iniţializate suprascriind valorile implicite, astfel:

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

În acest caz, secvenţa de elemente este forţată să pornească de la 1 în loc de 0.

Fiecare tip enumerare care este folosit are un tip_dată pentru elementele sale. Dacă nu se specifică nici un `tip_dată`, atunci se presupune implicit tipul `int`. Specificarea unui `tip_dată` (care poate fi orice tip exceptând tipul `char`, se face prin enunţarea `tipului_dată` după numele enumerării. Declararea unui tip enumerare este de forma:

```
enum [Nume_tip] [: Tip_dată]
{
[identificator1][=valoare],
...
[identificatorn][=valoare]}
```

Exemplu:

```
enum MyEnum : byte
{
 triunghi,
 cerc
}
```

Valoarea fiecărei variabile poate fi specificată explicit:

```
enum Values
{
 a = 45,
 b = 23,
 c = a + b
}
```

Exemplu:

```
using System;
namespace tipulEnum {
 class Program {
   enum lunileAnului
   {
    Ianuarie = 1,
    Februarie, Martie, Aprilie, Mai, Iunie, Iulie,
    August, Septembrie, Octombrie, Noiembrie, Decembrie
   }

 static void Main()
 {
   Console.WriteLine("Luna Mai este a {0}", (int)lunileAnului.Mai + "-a luna din 
an.");
```

```
  Console.WriteLine("Luna August este a {0}", (int)lunileAnului.August + "-a
luna din an.");
Valoarea lui i in acest ciclu este:Console.ReadLine();
  }
 }
}
```

De observat că un element din lista de enumerări poate fi accesat astfel:

`NumeEnum.NumeElement`

Tipurile enumerare pot fi convertite către tipul lor de bază şi înapoi, folosind o conversie explicită (cast):

```
enum Values
{
 a = 1,
 b = 5,
 c = 3
}
 class Test
 {
  public static void Main()
   {
   Values v = (Values)3;
   int ival = (int)v;
   }
 }
```

Exemplu

```
using System;
public class EnumTest
{
    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

    static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
        Console.ReadLine();
    }
}
```

4.2 Tipul tablou

Tipul tablou (Array) reprezintă o colecţie de valori de acelaşi tip. Tipul Array este un tip referinţă, fiecare array este un obiect moştenit din clasa de bază System.Array.

Tipul Array se declară astfel: `<tip_dată>[] <nume_variabilă>;`

Prin această declaraţie nu se alocă spaţiu pentru memorare.

Pentru a putea reţine date în structura de tip tablou, este necesară o operaţie de instanţiere:

`nume = new tip_dată[NumarElemente];`

```
}
```

Se reaminteşte că instanţierea este procesul de creare a unui obiect şi iniţializarea sa cu date specifice.

Declararea, instanţierea şi chiar iniţializarea tabloului se pot face în aceeaşi instrucţiune.
Exemplu:

```
int[] v = new int[] {1,2,3}; sau
int[] v = {1,2,3}; //new este implicit
```

Dimensiunea unui tablou trebuie fixată şi definită înaintea utilizării acestuia:

```
int size = 10;
int[] integers = new int[size];
```

Opţional, un tablou se poate declara şi iniţializa în paşi separaţi:

```
int[] integers;
integers = new int[10];
```

Exemplu de declarare şi iniţializare a unui vector cu cinci elemente:

```
int [] integers = {1, 2, 3, 4, 5};
```

Accesarea valorilor stocate într-un tablou.

Pentru a accesa elementele unui tablou, se foloseşte operatorul de indexare [int index]. Vom folosi acest index pentru a indica elementul din tablou pe care vrem sa-l accesăm. Este important de reţinut ca valoarea indexului în C# porneşte de la 0.

Următorul cod de program demonstrează cum poate fi accesat al treilea element al unui tablou:

```
int [] intArray = {5, 10, 15, 20};
int j = intArray[2];
```

Următorul cod de program prezintă declararea, iniţializarea şi parcurgerea unui tablou:

```
using System;
namespace UtilizareTablou
{
    class DemoArray
    {
        // demonstrează utilizarea arrays în C#
        static void Main()
        {
            // declararea şi iniţializarea unui array de întregi
            int[] integers = { 3, 7, 2, 14, 65 };
            // parcurgerea tabloului şi afişarea fiecărui element pe ecran
            for (int i = 0; i < 5; i++)
                Console.WriteLine(integers[i]);
                Console.ReadLine();
        }
    }
}
```

În cazul tablourilor cu mai multe dimensiuni se face distincţie între tablouri regulate şi tablouri neregulate (tablouri de tablouri).

Declararea în cazul tablourilor regulate bidimensionale se face astfel: `Tip_dată[,] nume;`
Instanţierea: `nume = new Tip_dată[Linii,Coloane];`

Acces: nume[indice1,indice2]

Exemple:
int[,] mat = new int[,] {{1,2,3},{4,5,6},{7,8,9}}; sau
int[,] mat = {{1,2,3},{4,5,6},{7,8,9}};

Declarare în cazul tablourilor neregulate bidimensionale: Tip[][] nume;

Intanţiere:
nume = new Tip[NrLinii],[];
nume[0]=new Tip[NrColoane1]
...
nume[NrLinii-1]=new Tip[NrColoaneLinii-1]

Acces: nume[indice1][indice2]

Exemple:
```
int[][] mat = new int[][] {
new int[3] {1,2,3},
new int[2] {4,5},
new int[4] {7,8,9,1}
  };
```
sau
```
int[][] mat={new int[3] {1,2,3},new int[2] {4,5},new int[4] {7,8,9,1}};
```

4.3 Conversii numerice.

În C# există două tipuri de conversii numerice:

- implicite
- explicite.

Conversia implicită se efectuează (automat) doar dacă nu este afectată valoarea convertită.
Exemplu:
```
using System;
namespace Conversii
{
    class Program
    {
        static void Main()
        {
            byte a = 13; // byte întreg fără semn pe 8 biţi
            byte b = 20;
            long c; //întreg cu semn pe 64 biţi
            c = a + b; // conversie
            Console.WriteLine(c);
        }
    }
}
```

Regulile de conversie implicită sunt descrise de tabelul următor:

Din	În
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double
ulong	float, double, decimal

Conversia explicită se realizează prin intermediul unei expresii **cast**, atunci când nu există posibilitatea unei conversii implicite.

În urma rulării programului:

```
using System;
namespace Conversii1
{
 class Program
 {
  static void Main(string[] args)
  {
  int a = 5;
  int b = 2;
  float c;
  c = (float)a / b; //operatorul cast
  Console.WriteLine("{0}/{1}={2}", a, b, c);
  Console.ReadLine();
  }
 }
}
```

se obţine:

5/2 = 2.5

În cazul în care nu s-ar fi folosit operatorul cast rezultatul, evident eronat, ar fi fost: 5/2=2.

Regulile de conversie explicită sunt descrise de tabelul următor:

Din	În
sbyte	byte, ushort, uint, ulong, char
byte	sbyte, char
short	sbyte, byte, ushort, uint, ulong, char
ushort	sbyte, byte, short, char
int	sbyte, byte, short, ushort, uint, ulong, char
uint	sbyte,byte, short, ushort, int, char
long	sbyte, byte, short, ushort, int, uint, ulong, char
ulong	sbyte, byte, short, ushort, int, uint, long, char
char	sbyte, byte, short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double

Conversii între numere şiruri de caractere.

În limbajul C# există posibilitatea efectuării de conversii între numere şiruri de caractere. Pentru conversia inversă, adică **din şir de caractere în număr**, sintaxa este:

```
şir →int int.Parse(şir) sau Int32.Parse(şir)
şir →long long.Parse(şir) sau Int64.Parse(şir)
şir →double double.Parse(şir) sau Double.Parse(şir)
şir →float float.Parse(şir) sau Float.Parse(şir)
```

Observaţie: În cazul în care şirul de caractere nu reprezintă un număr valid, conversia din şir în număr va eşua.

Exemplu:

```
using System;
namespace Conversii
{
 class Program
 {
  static void Main()
  {
      string s;
      const int a = 13;
      const long b = 100000;
      const float c = 2.15F;
      double d = 3.1415;
      Console.WriteLine("CONVERSII\n");
      Console.WriteLine("TIP\tVAL. \tSTRING");
      Console.WriteLine("———————————");
      s = "" + a;
      Console.WriteLine("int\t{0} \t{1}",a,s);
      s = "" + b;
      Console.WriteLine("long\t{0} \t{1}",b,s);
      s = "" + c;
      Console.WriteLine("float\t{0} \t{1}",c,s);
      s = "" + d;
      Console.WriteLine("double\t{0} \t{1}",d,s);
      Console.WriteLine("\nSTRING\tVAL \tTIP");
      Console.WriteLine("———————————");
      int a1;
      a1 = int.Parse("13");
      Console.WriteLine("{0}\t{1}\tint","13",a1);
      long b2;
      b2 = long.Parse("1000");
      Console.WriteLine("{0}\t{1}\tlong","1000",b2);
      float c2;
      c2 = float.Parse("2,15");
      Console.WriteLine("{0}\t{1}\tfloat","2,15",c2);
      double d2;
      d2 = double.Parse("3.1415");
      Console.WriteLine("{0}\t{1}\tdouble","3.1415",d2);
      Console.ReadLine();
  }
 }
}
```

CAPITOLUL V. CLASE – NOŢIUNI DE BAZĂ

5.1 Clasele. Noţiuni de bază

Tipul clasă (class) este cel mai important dintre tipurile definite de utilizator, în C#. Clasele sunt tipuri referinţă definite de utilizator şi reprezintă o mulţime încapsulată de date şi funcţii dependente logic, care în general modelează obiecte din lumea reală sau conceptuală.
Clasa grupează datele şi metodele (funcţiile) de prelucrare a acestora într-un modul, unindu-le astfel într-o entitate mult mai naturală. Deşi tehnica se numeşte "Programare Orientată Obiect", conceptul de baza al ei este „clasa”. Clasa, pe lângă faptul că abstractizează foarte mult analiza/sinteza problemei, are proprietatea de generalitate, ea desemnând o mulţime de obiecte care împart o serie de proprietăţi.

Clasa "floare", de exemplu, desemnează toate plantele care au flori, precum clasa "Fruct" desemnează toate obiectele pe care noi le identificam ca fiind fructe. Bineînţeles, în implementarea efectivă a programului nu se lucrează cu entităţi abstracte, precum clasele ci se lucrează cu obiecte, care sunt "instanţieri" ale claselor. Altfel spus, plecând de la exemplul de mai sus, dacă se construieşte un program care să lucreze cu clasa fructe, el nu va prelucra entitatea "fruct" ci va lucra cu entităţi concrete ale clasei "fruct", adică "afină", "cireaşă", "zmeură", etc.

Instanţierea (trecerea de la clasă la obiect) înseamnă atribuirea unor proprietăţi specifice clasei, astfel încât aceasta să indice un obiect anume, care se diferenţiază de toate celelalte obiecte din clasă printr-o serie de atribute. Dacă clasa „fruct” conţine caracteristicile: zonă, gust, şi culoare, atunci vom considera fructul "zmeură" se găseşte în zonele împădurite, de culoare roză şi gust dulce-acrişor. Fructul „zmeură” individualizează clasa „fruct”, astfel că ajungem la un caz concret, care este *obiectul*.

O *instanţă* a unui tip de date abstract este o ”concretizare” a tipului respectiv, formată din valori efective ale datelor. O instanţă a unui tip obiectual poartă numele de *obiect*.

Membrii unei clase sunt împărţiţi în următoarele categorii:
• constante
• câmpuri
• metode
• proprietăţi
• evenimente
• indexatori
• operatori
• constructori (de instanţă)
• destructor
• constructor static
• tipuri

O clasă este o structură de date care poate stoca date şi executa cod de program. În continuare sunt descrişi:

- *Membrii de tip dată (Data members),* care stochează date asociate clasei sau instanţe ale clasei. Membrii de tip dată, modelează, în general, atributele lumii reale, pe care clasa vrea să o reprezinte.
- *Membrii de tip funcţie (Function members)*, care execută cod de program. Membrii de tip funcţie modelează, în general, funcţii şi acţiuni ale obiectelor din lumea reală, pe care clasa vrea să le reprezinte.

Un program care rulează este o mulţime de obiecte care interacţionează unul cu celălalt.

Declararea claselor. Declararea unei clase defineşte caracteristicile şi metodele unei noi clase. Aceasta nu creează o instanţă a clasei, dar creează un şablon pe baza căruia se vor crea instanţele clasei. Următorul exemplu foloseşte sintaxa minimă pentru crearea unei clase:

cuvânt cheie   numele clasei

```
  ↓      ↓
class ExerciţiuClasă
{
DeclaraţiiDeMembrii
}
```

Între cele două acolade sunt declaraţi membrii clasei care formează, în acelaşi timp, corpul clasei. Membrii unei clase pot fi declaraţi în orice ordine în interiorul corpului clasei.

5.2 Câmpuri

Câmpurile(fields) şi metodele(methods) fac parte din tipurile cele mai importante de membrii ai unei clase.

Un **câmp** este o variabilă care aparţine unei clase. Acesta poate fi de orice tip: predefinit sau definit de utilizator. Asemenea variabilelor, câmpurile stochează date, şi au următoarele caracteristici:

Sintaxa minimală pentru declararea unui câmp este:

```
 tip
 ↓
Type Identifier;
     ↑
    Nume câmp
```

De exemplu, următoarea clasă conţine declaraţia câmpului CâmpulMeu, care poate stoca o valoare întreagă:

```
class MyClass
{       tip
         ↓
      int CâmpulMeu;
             ↑
}□□□□        □□□□□Nume tip
```

Iniţializarea explicită şi implicită a câmpurilor

Întrucât un câmp este un tip de variabilă, sintaxa pentru iniţializarea unui câmp este aceeaşi cu cea de la variabile.

```
class MyClass
{
int F1 = 17;
}       ↑
```
Iniţializarea câmpului

Pentru fiecare câmp declarat se va asigna o valoare implicită astfel:

- numeric: 0
- bool: false
- char: '\0'
- enum: 0
- referinţă: null

Exemplu:
```
class MyClass
{
int F1; // Iniţializat cu 0 - tip valoare
string F2; // Iniţializat cu null - tip referinţă
int F3 = 25; // Iniţializat cu 25
string F4 = "abcd"; // Iniţializat cu "abcd"
}
```

Acelaşi exemplu poate fi scris şi:
```
class MyClass
{
int F1, F3 = 25;
string F2, F4 = "abcd";
}
```

Un astfel de câmp se poate folosi fie prin specificarea numelui său, fie printr-o calificare bazată pe numele clasei sau al unui obiect.

Exemplu:
```
using System;
class Access
{
    //Variabile declarate înafara funcţiei Main.
    int x = 100;
    int y = 200;
    public static void Main()
    {
        //crearea obiectului
        Access a = new Access();

        //apelul variabilelor instanţiate
        Console.WriteLine(a.x);
        Console.WriteLine(a.y);
        Console.ReadLine();
    }
}
```

5.3 Metode

O metodă este un nume de bloc de cod executabil care poate fi executat din diferite părţi ale unui program, chiar şi din alte programe. Atunci când o metodă este apelată (invocată), aceasta

execută codul din corpul său, după care redă controlul mai departe programului din care a fost apelată. Unele metode returnează o valoare în poziţia în care au fost apelate. Metodele corespund funcţiilor membre din C++. Sintaxa minimă necesară declaraţiei unei metode include următoarele componente:

- Tipul returnat (*Return type*): tipul valorii returnate de metodă. În cazul în care metoda nu returnează o valoare, tipul returnat este **void**;
- Numele (name): reprezintă numele metodei;
- Lista parametrilor (*Parameter list*): aceasta constă din cel puţin un set de paranteze deschise „()”. Dacă există parametrii aceştia sunt prezentaţi între paranteze;
- Corpul metodei (Method body): conţine cod executabil inclus între două acolade „{}”.

Exemplu:

```
class SimpleClass
{
Tipul returnat  Lista de parametrii
 ↓              ↓
void PrintNums ( )
 {
  Console.WriteLine("1");
  Console.WriteLine("2");
 }
}
```

5.3.1 Structura unei metode

În esenţă o metodă este un bloc de cod care are un nume şi care poate fi apelat prin acesta. Se pot transmite date într-o metodă şi se pot prelua date de la o metodă. O metodă este un membru de tip funcţie al clasei în care este definită. Orice metodă este compusă din două părţi:

- antetul metodei în care se specifică caracteristicile metodei cum sunt:
  - dacă metoda returnează o valoare iar în caz afirmativ care este tipul acesteia,
  - numele metodei,
  - ce tip de date sunt transmise metodei.
- corpul metodei care conţine secvenţe de instrucţiuni (cod executabil). Execuţia instrucţiunilor porneşte de la prima şi se continuă secvenţial prin corpul metodei.

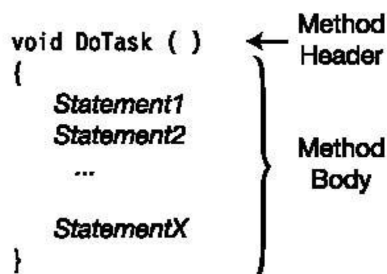


Figura 6. Structura unei metode

Structura antetului unei metode este:

```
int MyMethod ( int intpar1, string strpar1 )
 ↑      ↑                        ↑
```

Tipul returnat   Numele metodei   Lista de parametrii

5.3.2 Variabile locale

Asemenea câmpurilor, variabilele locale pot stoca date. În timp ce câmpurile unui obiect stochează, de obicei, starea unui obiect, variabilele locale sunt folosite pentru calcule locale sau tranzitorii.

Următoarele linii de cod reprezintă sintaxa declaraţiei unei variabile locale:

```
  Numele variabilei     Iniţializarea opţională
        ↓         ↓
Type Identifier = Value;
```

- existenţa unei variabile locale este limitată la blocul în care a fost declarată, şi la blocurile conţinute în acesta.
- variabilă locală poate fi declarată în orice poziţie în corpul unei metode.

Următorul exemplu ilustrează declararea şi utilizarea a două variabile locale. Prima este de tip întreg iar a doua de tip clasă SomeClass:

```
static void Main( )
{
int myInt = 15;
SomeClass sc = new SomeClass();
...
}
```

Următorul tabel face o comparaţie între variabilele locale şi instanţa câmpurilor.

	Instanţa unui câmp	**Variabila locală**
Timpul de viaţă	Există în momentul creării instanţei. Încetează să mai existe atunci când nu mai este accesată	Există din momentul în care este declarată. Încetează în momentul în care întregul bloc a fost executat.
Iniţializare implicită	Este iniţializată cu valoarea implicită a tipului	Nu sunt iniţializate implicit.
Zona de stocare	Toate câmpurile unei clase sunt stocate în memoria de tip heap, indiferent de tipul acestora.	Tipul valoare este stocat în memoria stack iar tipul referinţă este stocat în stack şi datele în heap.

5.3.3 Cuvântul cheie `var`

Începând cu versiunea C# 3.0 se poate utiliza cuvântul cheie `var` în locul declaraţiei tipului unei variabile locale, aşa cum se vede în exemplul următor:

```
static void Main( )
{
int total = 15;
MyExcellentClass mec = new MyExcellentClass();
...
}
```

este echivalent cu:

```
static void Main( )
{Keyword
 ↓
var total = 15;
var mec = new MyExcellentClass();
...
}
```

Câteva condiţii importante în utilizarea cuvântului cheie `var` sunt:

- poate fi folosit doar la declararea variabilelor locale – nu poate fi folosit la declararea câmpurilor unei clase;
- poate fi folosit numai atunci când declararea variabilelor locale include şi iniţializarea acestora;
- odată ce compilatorul a determinat tipul variabilelor, acesta nu mai poate fi schimbat.

Vizibilitatea variabilelor locale în interiorul blocurilor imbricate

Corpul metodelor poate conţine mai multe blocuri imbricate. Variabilele locale pot fi declarate în interiorul blocurilor imbricate, şi asemenea tuturor variabilelor locale, timpul acestora de viaţă este limitat la blocul în care au fost definite şi la blocurile conţinute în acesta.

Constantele locale

Ciclul de viaţă al constantelor locale este asigurat de aceleaşi reguli ca la variabilele locale. Sintaxa declaraţiei constantelor locale este:

```
cuvânt cheie
  ↓
const Type Identifier = Value;
                          ↑
             iniţializarea este obligatorie
```

5.3.4 Apelul unei metode

Se pot apela alte metode din corpul unei metode. O metodă se poate apela utilizând numele acesteia împreună cu lista de parametrii.

În următorul exemplu în clasa `MyClass` este declarată metoda `PrintDateAndTime` care este apelată în interiorul metodei `Main`.

```
class MyClass
{
  void PrintDateAndTime( )       // Declararea metodei.
 {
  DateTime dt = DateTime.Now;    // Det. datei şi a orei curente.
  Console.WriteLine("{0}", dt);  // Afişarea acesteia.
 }
static void Main()               // Declararea metodei principale.
 {
  MyClass mc = new MyClass();
  mc.PrintDateAndTime( );        // Apelarea metodei.
 }          ↑           ↑
}    numele metodei   lista vidă de parametrii
```

5.3.5 Valoarea returnată de o metodă

O metodă poate returna o valoare atunci când este apelată. Valoarea returnată este inserată în codul apelant în instrucţiunea în care este apelată metoda.

- Pentru a putea returna o valoare, trebuie declarat tipul acesteia înaintea numelui metodei (la declararea metodei);
- Dacă o metodă nu returnează o valoare, ea trebuie declarată de tip `void` (vid)

Următorul cod declară două metode. Prima returnează o valoare întreagă iar cea de-a doua o valoare vidă.

Tipul returnat

```
 ↓
int GetHour() { ... }
void DisplayHour() { ... }
 ↑
```

Nu este returnată nici o valoare.

O metodă care are returnează o valoare nevidă trebuie să conţină următoarea formă a instrucţiunii `return`:

```
return Expression; // Returnarea unei valori.
   ↑
```

Cuvânt cheie

Exemplu:

Tip returnat

```
 ↓
int GetHour( )
 {
 int a=1;
 int b=2;
 return a+b;
 } ↑
```

Instrucţiunea return

O metodă poate returna un obiect definit de programator. De exemplu, următorul cod returnează un obiect de tip `MyClass`.

Tipul returnat `-- MyClass`

```
  ↓
MyClass method3( )
  {
   MyClass mc = new MyClass();
   ...
   return mc; // returnează un obiect de tip MyClass.
  }
```

Instrucţiunea `return` folosită şi la metodele ce returnează tipul `void`.

Dintr-o metodă ce returnează tipul vid, se poate ieşi simplu prin utilizarea instrucţiunii return fără parametrii: `return;`

Această formă a instrucţiunii `return` poate fi folosită numai cu metodele declarate `void`. Execuţia instrucţiunilor din corpul unei metode declarate `void`, se opreşte atunci când este întâlnită instrucţiunea `return`.

Exemplu:

Tipul returnat

```
 ↓
void SomeMethod()
 {
  ...
  if ( Prima_condiţie ) // Dacă ...
  return; // controlul programului este returnat codului apelant.
  ...
  if ( A_doua_condiţie ) // Dacă ...
```

```
  return; // controlul programului este returnat codului apelant.
  ...
 } // controlul programului este returnat codului apelant.
```

Următorul cod este un alt exemplu de utilizare a instrucţiunii `return` într-o metodă declarată `void`:

```
class MyClass
{
Returnează tipul void
        ↓
      void TimeUpdate()
      {
       DateTime dt = DateTime.Now; // Obţinerea datei şi orei curente.
       if (dt.Hour < 12) // Dacă ora este mai mică decât 12,
          return; // atunci return.
            ↑
       Return la metoda apelantă.
       Console.WriteLine("It's afternoon!"); // Altfel, afişează mesaj.
      }
static void Main()
 {
  MyClass mc = new MyClass(); // Crează o instanţă a clasei.
  mc.TimeUpdate(); // Apelează metoda.
 }
}
```

5.3.6 Parametrii unei metode

Parametrii formali sunt variabile locale declarate in lista de parametrii a metodei. Următoarea declaraţie de metodă ilustrează sintaxa declaraţiei de parametrii:

```
public void PrintSum( int x, float y )
{ ... }                     ↑
Declararea parametrilor formali
```

- Deoarece parametrii formali sunt variabile locale, aceştia au un tip şi un nume.
- Parametrii formali sunt definiţi înafara corpului metodei şi iniţializaţi înaintea execuţiei corpului metodei, excepţie făcând un anumit tip de parametrii care vor fi discutaţi în capitolele următoare.
- Numărul parametrilor formali poate fi oricât, declaraţiile acestora fiind despărţite prin virgulă.
- Parametrii formali sunt folosiţi în corpul metodei, la fel ca şi variabilele locale. În exemplul următor, declaraţia metodei `PrintSum` foloseşte doi parametrii formali, `x`, `z` şi variabila locală `Sum`, toate de tip `int`.

```
public void PrintSum( int x, int y )
 {
  int Sum = x + y;
  Console.WriteLine("Newsflash: {0} + {1} is {2}", x, y, Sum);
 }
```

Parametrii actuali

Atunci când programul apelează o metodă, valorile parametrilor formali trebuie iniţializaţi înaintea execuţiei codului metodei.

- Expresiile sau variabilele (câmpurile) folosite la iniţializarea parametrilor formali se numesc parametrii actuali.
- Parametrii actuali sunt plasaţi în lista de parametrii ai metodei invocate.

Exemplul următor ilustrează invocarea metodei `PrintSum`, care are doi parametrii actuali de tip `int`:

```
PrintSum( 5, SomeInt );
          ↑     ↑
```

Expresie Variabilă de tip int

Atunci când o metodă este apelată, valoarea fiecărui parametru actual este utilizată pentru iniţializarea parametrului formal corespunzător. Figura următoare explică relaţia dintre parametrii actuali şi cei formali.

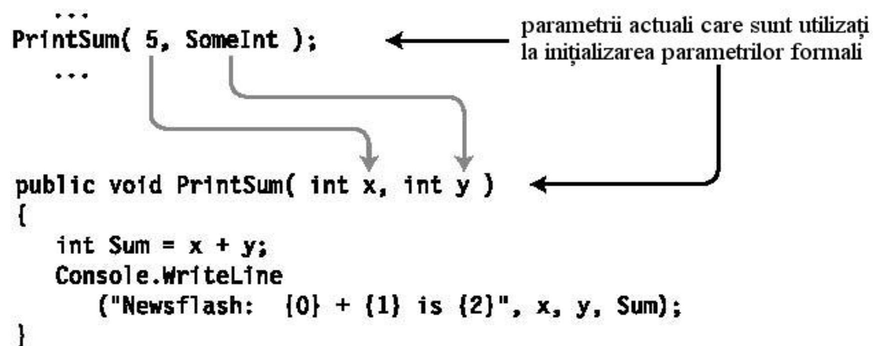


Figura 7. Relaţia dintre parametrii actuali şi cei formali

Atunci când este apelată o metodă, trebuie îndeplinite următoarele condiţii:

- Numărul parametrilor actuali trebuie să fie exact cu numărul parametrilor formali, cu o singură excepţie care va fi discutată în capitolele următoare.
- Fiecare parametru actual trebuie să aibă acelaşi tip cu parametrul formal corespunzător.

Exemple:

```
class MyClass                parametrii formali
{                                ↓      ↓
      public int Sum(int x, int y) // Declararea metodei.
      {
       return x + y; // Returnează suma.
      }
                                          parametrii formali
                                          ↓             ↓
      public float Avg(float Input1, float Input2) // Declararea metodei.
      {
       return (Input1 + Input2) / 2.0F; // Returnează media.
      }
}
```

```
class Class1
{
      static void Main()
      {
      MyClass MyT = new MyClass();
      int SomeInt = 6;
      Console.WriteLine("Newsflash: Sum: {0} and {1} is {2}",5,
SomeInt, MyT.Sum( 5,SomeInt )); // Invocarea metodei.
                         ↑
```

parametrii actuali

```
Console.WriteLine("Newsflash: Avg: {0} and {1} is {2}",5, SomeInt,
      MyT.Avg( 5, SomeInt )); // Invocarea metodei.
      }                  ↑
}
```

parametrii actuali

Codul de mai înainte produce următoarea ieşire:

```
Newsflash: Sum: 5 and 6 is 11
Newsflash: Avg: 5 and 6 is 5.5
```

Parametrii de tip valoare

Acesta este un tip implicit de parametri numiţi şi *parametrii valoare.* Atunci când se folosesc parametrii valoare, datele sunt transmise metodei prin copierea valorilor parametrilor actuali în parametrii formali. Atunci când o metodă este apelată, sistemul parcurge paşii următori:

- Alocă spaţiu în stivă (stack) pentru parametrii formali
- Copiază parametrii actuali în parametrii formali

Un parametru actual al unui parametru valoare poate fi, pe lângă variabilă locală, o expresie a cărui tip se potriveşte cu tipul parametrului valoare.

Exemplu:

```
float func1( float Val ) // Declararea metodei.
{ ... }          ↑
```

dată de tip float

```
      {
      float j = 2.6F;
      float k = 5.1F;
```

variabilă de tip float

```
      ↓
      float fValue1 = func1( k ); // Apelul metodei
      float fValue2 = func1( (k + j) / 3 ); // Apelul metodei
      ...                            ↑
```

Expresie care evaluează un tip real

Următorul cod de program prezintă o metodă denumită `MyMethod`, care are doi parametrii: o variabilă de tip `MyClass` şi o variabilă de tip `int`:

```
class MyClass
{
  public int Val = 20;  // Iniţializează câmpul Val cu 20.
}
```

```
class Program
                                             parametrii formali
{                                              ↓        ↓
      static void MyMethod( MyClass f1, int f2 )
      {
       f1.Val = f1.Val + 5; // Adună 5 la câmpul parametrului f1.
       f2 = f2 + 5; // Adună 5 la al doilea parametru.
      }
      static void Main( )
      {
      MyClass A1 = new MyClass();
      int A2 = 10;
      MyMethod( A1, A2 ); // Apelul metodei.
      }          ↑    ↑
}            parametrii actuali
```

Parametrii referință

Al doilea tip de parametrii folosiţi în lista de parametrii ai unei metode sunt parametrii referinţă.

- Atunci când se utilizează un parametru referinţă, trebuie folosit modificatorul ref atât în declaraţia metodei cât şi în invocarea acesteia.
- Parametrii actuali trebuie să fie variabile (nu sunt admise valori sau expresii valorice) care trebuie să fie iniţializate înainte să fie folosite ca parametrii actuali. Dacă parametrii actuali sunt variabile de tip referinţă, atunci acestora trebuie să li se atribuie, fiecăreia, o referinţă sau referinţa nulă (null).

Exemplul următor ilustrează sintaxa declarării şi a invocării unei metode cu parametrii referinţă.

```
    include modificatorul ref
                ↓
void MyMethod( ref int val ) // Declararea metodei
{ ... }
int y = 1; // Variabila pentru parametrul actual
MyMethod ( ref y ); // Apelul metodei
            ↑
  include modificatorul ref
MyMethod ( ref 3+5 ); // Error!
                ↑
    trebuie utilizată o variabilă
```

Aşa cum am văzut, în cazul parametrilor valoare sistemul alocă memorie în stivă (stack) pentru parametrii formali. Pentru parametrii referinţă există următoarele caracteristici:

- Nu se alocă memorie în stivă pentru parametrii formali.
- În schimb, numele parametrilor formali acţionează ca un alias pentru variabilele considerate parametrii actuali, referind aceeaşi locaţie de memorie.

Întrucât numele parametrilor formali şi numele parametrilor actuali referă aceeaşi locaţie de memorie, este evident că orice schimbare făcută parametrilor formali, în timpul execuţiei codului metodei respective, va avea efect după execuţia metodei, asupra variabilelor desemnate ca parametrii actuali.

Exemplu:

```
class MyClass
{ public int Val = 20; } // Iniţializarea câmpului cu valoarea 20.
class Program
 {
```

modificatorul `ref` **modificatorul** `ref`

```
                            ↓                ↓
      static void MyMethod(ref MyClass f1, ref int f2)
      {
       f1.Val = f1.Val + 5; // Adună 5 la câmpul parametrului f1.
       f2 = f2 + 5; // Adună 5 la al doilea parametru.
      }
      static void Main()
      {
       MyClass A1 = new MyClass();
       int A2 = 10;
       MyMethod(ref A1, ref A2); // Apelul metodei.
 }                 ↑       ↑
}
```

modificatori `ref`

Parametrii de ieşire

Parametrii de ieşire sunt folosiţi pentru a transmite date dinăuntrul unui metode, în codul apelant. Parametrii de ieşire sunt asemănători cu parametrii referinţă, şi au următoarele caracteristici:

- Atunci când se utilizează un parametru de ieşire, trebuie folosit modificatorul `out` atât în declaraţia metodei cât şi în invocarea acesteia.
- Parametrii actuali trebuie să fie variabile (nu sunt admise valori sau expresii valorice).

În exemplul următor este declarată metoda `MyMethod`, care are un singur parametru de ieşire:

modificatorul `out`

```
                 ↓
void MyMethod( out int val ) // Declaraţia metodei
{ ... }
...
int y = 1; // variabila utilizată ca parametru actual
MyMethod ( out y ); // Apelul metodei
            ↑
```

modificatorul `out`

Asemenea parametrilor referinţă, parametrii formali ai parametrilor de ieşire acţionează ca aliasuri pentru parametrii actuali. Numele parametrilor formali şi numele parametrilor actuali referă, fiecare în parte, aceeaşi locaţie de memorie. Este evident faptul că orice acţiune asupra parametrilor formali făcută în interiorul corpului metodei, va avea acelaşi efect asupra parametrilor actuali. Spre deosebire de parametrii referinţă, parametrii de ieşire au următoarele caracteristici:

- Unui parametru de ieşire trebuie să i se atribuie o valoare, în corpul metodei, înainte ca acesta să fie folosit. Aceasta înseamnă că valoarea iniţială a parametrilor actuali este irelevantă şi, din această cauză, nu trebuie atribuite valori parametrilor actuali înaintea apelului metodei.

- Fiecare parametru de ieşire trebuie iniţializat.

Deoarece parametrii de ieşire trebuie iniţializaţi în corpul metodei, este inutil să fie transmise date (prin parametrii actuali) în corpul metodei. Utilizarea unui parametru de ieşire înaintea iniţializării acestuia produce eroare.

Exemplu:
```
public void Add2( out int outValue )
  {
   int v1 = outValue + 2; //Eroare!Utilizarea unui param de ieşire
      înaintea iniţializării acestuia produce eroare.
  }
```

Exemplu:
```
class MyClass
{ public int Val = 20; } // Iniţializarea câmpului cu valoarea 20.

class Program {
    static void MyMethod(out MyClass f1, out int f2, out int f3){
        f1 = new MyClass(); // Crearea unui obiect al clasei.
        f1.Val = 25; // Atribuirea unui valori câmpului clasei.
        // f2 = f2 + 5; // Eroare !!
        f2 = 15;
        // eroare ! parametrul f3 nu este initializat
    }
    static void Main() {
        MyClass A1 = null;
        int A2, A3;
        MyMethod(out A1, out A2, out A3); // Apelul metodei.
    }
}
```

Parametrii de tip vector

Caracteristicile importante ale parametrilor de tip vector sunt:

- Poate exista numai un parametru de tip vector în lista de parametrii.
- Dacă există unul, acesta trebuie să fie ultimul parametru din listă.

Pentru a declara un parametru de tip vector trebuie:

- Utilizat modificatorul params înaintea tipului de date.
- Trebuie plasat un set de paranteze drepte goale după tipul de date.

Antetul metodei prezentate în continuare prezintă sintaxa pentru declararea unui parametru de tip vector de întregi.

vector de întregi
```
                          ↓
void ListInts( params int[] inVals )
{ ...              ↑            ↑
```
modificator **numele parametrului**
```
}
```

Apelul unei metode ce conţine parametrii de tip vector.

Un parametru actual de tip vector poate fi transmis unei metode în două moduri:

1. Printr-o listă de elemente de tipul specificat în declararea metodei, separate prin virgulă: `ListInts( 10, 20, 30 );` Această formă se mai numeşte şi forma expandată.

2. Printr-o variabilă de tip vector:

```
int[] intArray = {1, 2, 3};
ListInts( intArray );
```

Aşa cum se observă din exemplele date, la apelul metodei nu se utilizează modificatorul `params`.

În exemplul următor se observă că apelul metodei `ListInts` poate fi făcut cu un număr variabil de elemente.

```
void ListInts( params int[] inVals ) { ... } // Declararea metodei
...
ListInts( ); // 0 parametrii actuali
ListInts( 1, 2, 3 ); // 3 parametrii actuali
ListInts( 4, 5, 6, 7 ); // 4 parametrii actuali
ListInts( 8, 9, 10, 11, 12 ); // 5 parametrii actuali
```

Atunci când se apelează o metodă cu parametru vector în forma expandată, compilatorul face următoarele lucruri:

- Ia lista parametrilor actuali şi o utilizează la crearea şi iniţializarea unui vector în memoria heap.
- Memorează referinţa către vector, în stivă în locaţia parametrului formal.
- Dacă nu există nu există parametrii actuali în poziţia corespunzătoare parametrului formal de tip vector, compilatorul creează un vector cu zero elemente.

Exemplu:

```
class MyClass                    parametrii de tip vector
{                                      ↓
 public void ListInts( params int[] inVals )
      {
       if ( (inVals != null) && (inVals.Length != 0))
       for (int i = 0; i < inVals.Length; i++)
        {
         inVals[i] = inVals[i] * 10;
         Console.WriteLine("{0} ", inVals[i]);
        }
      }
}
class Program
{
      static void Main()
        {
        int first = 5, second = 6, third = 7;
        MyClass mc = new MyClass();
        mc.ListInts( first, second, third );
                                   ↑
                          parametrii actuali
        Console.WriteLine("{0}, {1}, {2}", first, second, third);
      }
}
```

Tabel cu sumarul tipurilor de parametrii

Tipul parametrului	**Modificator**	**Utilizat la declarare**	**Utilizat la invocare**	**Implementare**
Valoare	Nu			Compilatorul copiază parametrii actuali in parametrii formali
Referinţă	ref	Da	Da	Parametrii formali devin alias ai parametrilor actuali
De ieşire	out	Da	Da	Parametrii formali devin alias ai parametrilor actuali
De tip vector	params	Da	Nu	Este permisă transmiterea către metodă a unui număr variabil de parametrii actuali.

5.4 Crearea variabilelor şi instanţelor unei clase

Declararea unei clase este un şablon din care instanţele clasei sunt create.

- Clasele sunt tipuri referinţă, şi prin urmare necesită memorie atât pentru referinţa la date cât şi pentru datele actuale.
- Referinţa la date este stocată într-o variabilă de tip class. Astfel, pentru a crea o instanţă a clasei, trebuie, pentru început, declarată o variabilă de tip clasă. Dacă variabila nu este iniţializată, valoarea sa este nedefinită.

Alocarea memoriei pentru date

Declararea unei variabile de tip clasă alocă memorie pentru stocarea referinţei, nu şi pentru datele actuale ale obiectului instanţiat din clasă. Pentru a aloca memorie datelor actuale, trebuie utilizat operatorul de instanţiere new.

Operatorul `new` alocă şi iniţializează memorie pentru o instanţă a unui tip specificat. Acest operator alocă memorie din memoria de tip stack şi memoria de tip heap, în funcţie de tipul variabilei. Utilizarea operatorului de instanţiere new pentru crearea unui obiect constă din:

- Numele variabilei de tip clasă;
- Semnul „=”;
- Cuvântul cheie `new`;
- Numele tipului de instanţă pentru care se alocă memorie;
- Parantezele deschise care pot sau nu să conţină parametrii.

```
Cuvânt cheie                    paranteze deschise
 ↓                 ↓
Nume_variabilă = new TypeName( )
                ↑
                      numele tipului
```

Dacă alocarea memoriei este pentru un tip referinţă, expresia de creare a unui obiect returnează o referinţă către locaţia de memorie de tip heap, unde este alocată şi iniţializată instanţa.

```
Dealer theDealer;             // Declararea variabilei pentru referinţă.
theDealer = new Dealer();   // Alocarea memoriei pentru clasa obiect.
           ↑
        Expresia de creare a unui obiect
```

Declararea unei variabile de tip clasă şi iniţializarea ei se pot face într-o singură declaraţie:

Declararea variabilei

↓

```
Dealer theDealer = new Dealer(); // Declarare şi iniţializare (instanţierea).
```

↑

Iniţializare (instanţiere) cu sintaxa de creare a unui obiect

5.5 Membrii unei instanţe

Declararea unei clase produce un şablon din care se vor putea crea instanţe ale clasei respective.

Membrii instanţei (*Instance members*): Fiecare instanţă a unei clase este o entitate separată care are propriul set de *date membre*, distincte faţă de altă instanţă a aceleaşi clase. Acestea (datele membre) sunt denumite **membrii instanţei** (*instance members*), deoarece sunt asociaţi instanţei unei clase. Următorul exemplu ilustrează un program cu trei instanţe ale clasei `Player`.

```
class Dealer { ... }   //declararea clasei Dealer
class Player {            //declararea clasei Player
      string Name;       // câmp
      ...
}
class Program {
      static void Main()
       {
 Dealer theDealer = new Dealer();
 Player player1 = new Player();
 Player player2 = new Player();
 Player player3 = new Player();
       ...
       }
}
```

5.6 Modificatori de acces

Din interiorul unei clase, orice funcţie membru poate accesa oricare alt membru al clasei, simplu prin numele membrului.

Modificatorul de acces (*access modifier*) este o parte opţională din declaraţia unui membru, care specifică care părţi din program au acces la membru. Modificatorul de acces este plasat înaintea declaraţiei unui membru. Următoarele sintaxe sunt folosite pentru câmpuri şi metode:

Fields:

```
AccessModifier Type Identifier;
```

Methods:

```
AccessModifier ReturnType MethodName ()
{
...
}
```

Acestor membri li se pot ataşa următorii cinci modificatorii de acces:

- private
- public
- protected
- internal
- protected internal

5.7 Accesul privat sau public

Membrii care au specificatorul de acces `private` sunt accesibili doar în clasa în care au fost declaraţi – alte clase nu au acces la ei. Accesul de tip `private` este specificatorul implicit. Astfel, dacă un membru este declarat fără un modificator de acces, acesta este un membru privat.

De exemplu, următoarele două declaraţii specifică membrii privaţi de tip întreg:

```
int MyInt1;              // Declarare implicită
private int MyInt2;      // Declarare explicită
    ↑
```

Modificator de acces

Membrii care au specificatorul de acces public sunt accesibili tuturor obiectelor din program.

Modificator de acces

```
  ↓
public int MyInt;
```

Exemplu de accesare a membrilor unei clase:

```
class C1
{
      int F1;                 // Câmp implicit privat
      private int F2;         // Câmp declarat explicit privat
      public int F3;          // Câmp declarat public
      void DoCalc()           // Metodă implicit privată
      {
      ...
      }
      public int GetVal()  // Metodă declarată publică
      {
      ...
      }
}
```

Exemplu:

```
using System;
class Access
{
    //Variabile declarate înafara funcţiei Main.
    int x = 100;
    int y = 200;
}
class program
{
    public static void Main()
    {
        //Crearea unui obiect
```

```
        Access a = new Access();
        //Apelul instantei variabilelor
        Console.WriteLine(a.x); // Eroare, x nu e declarată public
        Console.WriteLine(a.y); // Eroare, y nu e declarată public
        Console.ReadLine();
    }
}
```

Accesarea membrilor din interiorul unei clase

Membrii unei clase pot fi accesaţi din interiorul acesteia utilizând numele lor.

În următorul exemplu metodele unei clase accesează câmpuri şi alte metode ale aceleiaşi clase:

```
class DaysTemp
{
// Câmpuri
      private int High = 75;
      private int Low = 45;
// Metode
private int GetHigh()
{
      return High; // Access private field
}
private int GetLow()
{
      return Low; // Access private field
}
public float Average ()
{
      return (GetHigh() + GetLow()) / 2; // Accesarea metodelor private
}               ↑            ↑
}             Accesarea metodelor private
```

Accesarea membrilor unei clase din afara acesteia

Pentru a accesa din afara unei clase membrii publici ai acesteia, aceştia trebuie apelaţi cu numele variabilei de tip clasă separat cu punct de numele membrului.

```
DaysTemp myDt = new DaysTemp(); // Crearea unui obiect al clasei.
float fValue = myDt.Average(); // Accesul din afara clasei.
         ↑   ↑
    numele variabilei   numele membrului
```

Următorul cod declară două clase: `DaysTemp` şi `Program`. Două câmpuri ale clasei `DaysTemp` sunt declarate public, prin urmare acestea vor putea fi accesate din afara clasei. În corpul metodei Main sunt create o variabilă şi un obiect al clasei `DaysTemp`, după care sunt atribuite valori câmpurilor obiectului.

```
using System;
class DaysTemp // declararea clasei DaysTemp
{
    public int High = 75;
    public int Low = 45;
}
class Program // declararea clasei Program.
{
    static void Main()
```

```
    {
    DaysTemp temp = new DaysTemp(); // crearea unui obiect.
    temp.High = 85; // atribuire de valori câmpurilor.
    temp.Low = 60;
    Console.WriteLine("High: {0}", temp.High ); // citirea valorii unui câmp.
    Console.WriteLine("Low: {0}", temp.Low );
    Console.ReadLine();
    }
}
```

Următorul cod creează două instanţe şi stochează referinţa acestora în variabilele cu numele t1 şi t2:

```
using System;
class DaysTemp // declararea unei clase.
{
    public int High, Low; // declararea câmpurilor.
    public int Average()  // declararea unei metode.
    {
    return (High + Low) / 2;
    }
}
class Program
{
static void Main()
    {
    DaysTemp t1 = new DaysTemp();
    DaysTemp t2 = new DaysTemp();
    // scriere de valori în câmpurile fiecărei instanţe.
    t1.High = 76; t1.Low = 57;
    t2.High = 75; t2.Low = 53;
    // citirea valorilor cîmpurilor fiecărei instanţe
    Console.WriteLine("t1: {0}, {1}, {2}", t1.High, t1.Low, t1.Average() );
    Console.WriteLine("t2: {0}, {1}, {2}", t2.High, t2.Low, t2.Average() );
    }
}
```

CAPITOLUL VI. PROBLEME REZOLVATE

6.1 ALGORITMI ELEMENTARI

1. *Interschimbaţi conţinutul a două numere de tip întreg citite de la tastatură.*

Program

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            int a, b, aux;
            Console.Write("Introduceti primul număr a= ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr b= ");
            b = int.Parse(Console.ReadLine());
            Console.WriteLine("a={0}, b={1}", a, b);
            aux = a; a = b; b = aux;
            //o alta versiune de interschimbare fara variabila suplimentara
            // a = a - b; b = a + b; a = b - a;
            Console.Write("Dupa interschimbare a={0}, b={1}",a,b);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

- pe prima linie în corpul funcţiei principale se declară trei variabile întregi: *a*, *b* şi *aux* ;
- se foloseste de două ori perechea de funcţii *Console.Write()/ int.Parse(Console.ReadLine())* pentru a afişa un mesaj pe ecran şi apoi pentru a citi cele două valori întregi de la tastatură, care vor fi depuse în variabilele *a*, respectiv *b*; deoarece numerele citite de la tastatură se introduc în valori de tip *int*, funcţia de citire de la tastatură *Console.ReadLine()* trebuie însoţită de transformarea intrării de tip String în *int* cu ajutorul funcţiei *int.Parse()*;
- urmează trei instrucţiuni de atribuire, care împreună formează „metoda celor trei pahare";
- se afişează rezultatul, folosind un apel de funcţie *Console.Write()*; observăm că se va tipări pe ecran atât şir de caractere cât şi două secvenţe de tipul {…}, care la tipărire se vor înlocui cu valorile din variabilele *a*, respectiv *b*;
- urmează un apel al funcţiei *Console.ReadKey()* care va ţine ecranul de Output vizibil până la apăsarea unei taste. În lipsa ei, acesta dispare imediat după afişarea rezultatelor.

Efect

Primul număr: 15

Al doilea număr: 21

După interschimbare: a=21 b=15

2. *Să se rezolve ecuaţia de gradul I de forma ax+b=0, cu coeficienţi numere reale.*

Program

```
using System;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main()
        {
            float a, b, x;
            Console.Write("Introduceti a=");
            a = float.Parse(Console.ReadLine());
            Console.Write("Introduceti b=");
            b = float.Parse(Console.ReadLine());
            if (a == 0)
                if (b == 0) Console.WriteLine("Ecuatie nedeterm");
                else
                    Console.WriteLine("Ecuatie imposibila");
            else
                {
                    x = - b / a;
                    Console.WriteLine("Solutia este x={0}",x);
                }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

În vederea realizării unei rezolvări riguroase, înainte de a afla soluţia banală x=-b/a, trebuiesc făcute nişte teste asupra valorilor a şi b. Astfel dacă a=0 şi b=0 avem de a face cu o ecuaţie nedeterminată, iar dacă a=0 şi b≠0 avem de a face cu o ecuaţie imposibilă. Acestea sunt cazuri particulare care trebuiesc tratate. În cazul în care a≠0 se poate extrage soluţia după formula cunoscută x=-b/a, iar această soluţie este tipărită şi pe ecran cu ajutorul funcţiei *Console.WriteLine()*. Pentru a avea instrumentele necesare calculului, în primă fază se vor citi de la tastatură valorile reale *a* şi *b*, după care se trece la găsirea soluţiei. Se observă ca pentru raţionamentul enunţat mai sus, au fost folosite în program două instrucţiuni *if-else* cu ajutorul cărora se testează valorile lui *a* şi *b*.

3. *Să se rezolve o ecuaţie de gradul II de forma* $ax^2+bx+c=0$*, cu coeficienţi numere reale.*

Program

```
using System;
namespace ConsoleApplication3
{
    class Program
    {
        static void Main()
        {
            float a, b, c, delta;
            double x1, x2;
```

```
            Console.Write("Introduceti a=");
            a = float.Parse(Console.ReadLine());
            Console.Write("Introduceti b=");
            b = float.Parse(Console.ReadLine());
            Console.Write("Introduceti c=");
            c = float.Parse(Console.ReadLine());
            if (a == 0)
                if (b == 0)
                    if (c==0) Console.WriteLine("Ecuatie nedet.");
                    else Console.WriteLine("Ecuatie imposibila");
                else
                {
                    x1 = -c / b;
                    Console.WriteLine("Ecuatia este de gradul 1 cu x1={0}", x1);
                }
            else
            {
                delta = b * b - 4 * a * c;
                if (delta < 0)
                    Console.WriteLine("Ec. are solutii complexe");
                else
                {
                    if (delta == 0)
                    {
                        x1 = x2 = -b / (2 * a);
                        Console.WriteLine ("x1=x2={0}", x1);
                    }
                    else
                    {
                        x1 = (-b + Math.Sqrt(delta)) / (2 * a);
                        x2 = (-b - Math.Sqrt(delta)) / (2 * a);
                        Console.WriteLine("Solutiile sunt x1={0}, x2={1}", x1,
x2);
                    }
                }
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

În vederea realizării unei rezolvări riguroase, înainte de a afla soluţia banală dată de formula $x_{1,2}=(-b\pm\sqrt{\Delta})/(2*a)$, unde $\Delta=b^2-4ac$, trebuiesc făcute nişte teste asupra valorilor a, b, c.

Se detectează următoarele situaţii:

- dacă a=0, b=0, c=0 avem de-a face cu o ecuaţie nedeterminată
- dacă a=0, b=0, c≠0 aveam de-a face cu o ecuaţie imposibilă
- dacă a=0, b≠0, iar c are orice valoare, avem de-a face cu o ecuaţie de gradul I, situaţie în care există o singură soluţie x=-c/b

Acestea sunt cazuri particulare care trebuiesc tratate.

- dacă a≠0, iar b şi c au orice valoare, avem de-a face cu o ecuaţie de gradul II, situaţie în care se poate trece la calculul lui Δ. Urmează acum o altă discuţie după valoarea lui Δ.

Astfel: - dacă $\Delta<0$ avem de-a face cu soluţii complexe (pe care în această rezolvare nu le vom mai calcula ci vom afişa doar un mesaj)

- dacă $\Delta=0$ ecuaţia are două soluţii egale: $x_{1,2}$=-b/(2*a)
- dacă $\Delta>0$ ecuaţia are două soluţii diferite:

$x_{1,2}=(-b\pm\sqrt{\Delta})/(2*a)$

Se observă că pentru raţionamentul enunţat mai sus, au fost folosite în program trei instrucţiuni *if-else* imbricate (*if* inclus în alt *if*) cu ajutorul cărora se testează valorile lui a, b şi c.

Puţină atenţie trebuie acordată expresiilor -b/(2*a) şi $(-b\pm\sqrt{\Delta})/(2*a)$ deoarece lipsa parantezelor din aceste expresii poate duce la rezultate greşite. Erorile pot apărea din cauză că în aceste expresii apar operatorii * şi / care au aceeaşi prioritate. Astfel, în lipsa vreunei paranteze ei s-ar executa în ordinea în care apar în expresie, lucru care nu este de dorit în acest caz.

4. *Scrieţi un program care primeşte la intrare un număr de secunde şi întoarce numărul maxim de ore, de minute, de secunde care este echivalent ca timp.*
Exemplu: 7384 secunde este echivalent cu 2 ore, 3 minute şi 4 secunde.

Program

```
using System;
namespace ConsoleApplication4
{
    class Program
    {
        static void Main()
        {
            int secunde, h, m, s;
            Console.Write("Introduceti numărul de secunde : ");
            secunde = int.Parse(Console.ReadLine());
            m = secunde / 60;
            s = secunde % 60;
            h = m / 60;
            m = m % 60;
            Console.Write("{0} secunde",secunde);
            Console.Write("reprez {0} ore, {1} minute şi {2} secunde", h, m, s);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Această problemă este una simplă care implică câteva calcule. Astfel, în prima fază se calculează câte minute reprezintă secundele date de problemă. Restul împărţirii secundelor iniţiale la 60 reprezintă câte secunde nu pot forma un minut întreg (s). Minutele obţinute se împart şi ele la 60 pentru a afla câte ore reprezintă acele minute (h). Restul împărţirii minutelor la 60 reprezintă câte minute nu pot forma o oră întreagă (m).

5. *Scrieţi un program care simulează un calculator electronic pentru numere întregi: se introduc două numere întregi şi o operaţie care poate fi +, -, *, /, reprezentând adunarea, scăderea, înmulţirea şi câtul.*

Program

```
using System;
namespace ConsoleApplication5 {
```

```
    class Program
    {
        static void Main()
        {
            int a,b,rez = 0;
            char op;
            short ok = 1;
            Console.Write("Primul număr : ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Al doilea număr : ");
            b = int.Parse(Console.ReadLine());
            Console.Write("Operatia dorita (+  -  *  /) : ");
            op = char.Parse(Console.ReadLine());
            switch (op)
            {
                case '+': rez = a + b; break;
                case '-': rez = a - b; break;
                case '*': rez = a * b; break;
                case '/':
                    if (b != 0) rez = a / b;
                    else
                    {
                        Console.WriteLine("Impartire la 0!");
                        Console.ReadKey();
                        return;
                    }
                    break;
                default: ok = 0; break;
            }
            if (ok == 1) Console.WriteLine("{0} {1} {2} = {3}", a, op, b, rez);
            else Console.WriteLine("Operator invalid");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

- se declară numerele întregi *a*, *b* şi *rez* de tip *int*, reprezentând operanzii şi rezultatul;
- se declară caracterul *op* de tip *char*, reprezentând operaţia care se va executa;
- se declară variabila *ok* de tip *short*, care va rămâne l dacă operaţia se termină cu succes, altfel i se va atribui 0 ;
- având în vedere ca va trebui să comparăm valoarea reţinută de variabila *op* cu mai multe valori, folosim instrucţiunea de selecţie *switch*. Avem 4 ramuri *case*, una pentru fiecare din valorile +, -, *, / şi ramura *default* pentru cazurile în care op are orice altă valoare înafara celor 4 enumerate anterior, moment în care ok ia valoarea 0.

În cazul în care *op* are una din valorile +, -, *, se efectuează operaţia corespunzătoare, iar rezultatul se depune în variabila *rez*. În cazul în care *op* are valoarea /, prima dată se va testa cu ajutorul unei instrucţiuni *if*, valoarea celui de-al doilea operand, deoarece dacă el este 0, împărţirea nu se poate realiza, caz în care se va afişa pe ecran un mesaj corespunzător. Dacă valoarea lui *b* este diferită de 0, se procedează ca şi în cazul operaţiilor +, -, *.

La final, în cadrul unei instrucţiuni *if* se testează valoarea variabilei *ok*. Dacă *ok* a rămas l, atunci se va afişa operaţia efectuată împreuna cu rezultatul, altfel se va tipări un mesaj că operatorul

introdus de la tastatură nu este unul valid.

Efect

Caz 1:
Primul număr: 15
Al doilea număr: 27
Operatia dorita (+ - * /): +
15 + 27 = 42
Caz 2:
Primul număr: 23
Al doilea număr: 0
Operatia dorita (+, -, *, /): /
Impartire la 0 !
Caz 3:
Primul număr: 38
Al doilea număr: 2
Operatia dorita (+, -, *, /): &
Operator invalid!

6. *Înmulţirea a două numere naturale prin adunări repetate.*

Program

```
using System;
namespace ConsoleApplication6
{
    class Program
    {
        static void Main()
        {
            int a, b, produs=0;
            Console.Write("Introduceti a=");
            a = int.Parse(Console.ReadLine());
            Console.Write("Introduceti b=");
            b = int.Parse(Console.ReadLine());
            for (int i = 1; i <= b; i++)
                produs += a;
            Console.WriteLine("{0} * {1} = {2}", a, b, produs);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se ştie încă din clasele elementare că înmulţirea înseamnă de fapt nişte adunări repetate. Astfel, a*b înseamnă *a adunat de b ori* sau *b adunat de a ori*.

Deoarece se ştie că *a* trebuie adunat de exact *b* ori pentru a se obţine rezultatul dorit, se utilizează instrucţiunea *for* care are un număr determinat de paşi (de câte ori se repetă un set de instrucţiuni). Valoarea *a* se adună se *b* ori în variabila *p,* care la final va conţine rezultatul căutat (*a*b*). Variabila *produs* este iniţializată cu 0 după care i se va adăuga în cadrul instrucţiunii *for* câte un *a*.

7. *Împărţirea a două numere prin scăderi repetate.*

Program

```
using System;
namespace ConsoleApplication7
{
    class Program
    {
        static void Main()
        {
            int a, b, a1 = 0, c=0, r=0;
            Console.Write("Introduceti a=");
            a = int.Parse(Console.ReadLine());
            Console.Write("Introduceti b=");
            b = int.Parse(Console.ReadLine());
            a1 = a;
            while (a1 >= b)
            {
                a1 -= b; //a1=a1-b;
                c++;
            }
            r = a1;
            Console.WriteLine("{0}:{1}={2} rest {3}", a, b, c, r);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se ştie încă din clasele elementare că împărţirea înseamnă de fapt nişte scăderi repetate. Astfel, a:b înseamnă *b scăzut de c ori* şi este posibil să existe şi un *rest*∈[0, b-1]. Nu se ştie de câte ori se va scădea *b* din *a*, acest lucru se va afla doar la final, de asemenea şi eventualul rest. În acest sens, pentru a repeta scăderea se va utiliza o instrucţiune repetitivă cu număr necunoscut de paşi şi anume *while*. Condiţia ca scăderea să se mai repete este ca *a>=b*. De fiecare dată când se mai face o scădere se va incrementa variabila *c*, care în final va reprezenta câtul. Valoarea care rămâne în final în *a* reprezintă restul.

Variabilele *c* şi *r* se iniţializează cu 0, urmând ca la final să conţină câtul (obţinut în cadrul ciclului *while*) şi restul împărţirii. Deoarece valoarea lui *a* se alterează în timpul calculului, dacă se doreşte se poate face o copie a sa înainte de a intra în ciclul *while*.

8. *Să se ghicească un număr întreg din intervalul 1 – 100.*

Program

```
using System;
namespace ConsoleApplication8
{
    class Program
    {
        static void Main()
        {
            Random sol = new Random();//generam un număr
            int solutie = sol.Next(100);//aleator ca şi solutie
            int n;
```

```
            do
            {
                Console.Write("Dati un număr intre 0 şi 100: ");
                n = int.Parse(Console.ReadLine());
                if (n < solutie)
                    Console.WriteLine("Numărul e prea mic!");
                else if (n == solutie)
                {
                    Console.WriteLine();
                    Console.WriteLine("BRAVO! Ati ghicit!");
                }
                else Console.WriteLine("Numărul e prea mare!");
            }
            while (n != solutie);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Acesta este un exemplu de utilizare a instrucţiunii repetitive *do-while* cu număr nedeterminat de paşi. Se setează din program o valoare din intervalul 0-100 (în variabila *soluţie*) care urmează să fie ghicită de către utilizator pe baza indicaţiilor *mai mare* sau *mai mic* pe care le va primi. Pentru generarea numerelor aleatoare s-a creat un obiect *sol* din clasa *Random* după care s-a generat un număr aleator din intervalul 0-100 cu ajutorul metodei *Next*, valoare care a fost atribuită variabilei *soluţie*.

Atâta timp cât soluţia nu este ghicită, dacă se introduce de la tastatură o valoare mai mică decât soluţia se va afişa mesajul *"Numărul e prea mic",* altfel se va afişa mesajul *"Numărul e prea mare"*, iar dacă se ghiceşte se va afişa mesajul *"BRAVO! Ati ghicit !"* şi programul se încheie.

9. *Să se calculeze n!=1*2*3*…*n (factorialul lui n), pentru un n natural citit de la tastatură.*

Program

```
using System;
namespace ConsoleApplication9
{
    class Program
    {
        static void Main()
        {
            short n, i;
            long fact=1;
            Console.Write("Introduceti n=");
            n=int.Parse(Console.ReadLine());
            for (i = 1; i <= n; i++)
                fact *= i; //fact = fact * i;
            Console.WriteLine("{0}! = {1}", n, fact);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

n! înseamnă înmulţirea tuturor numerelor naturale de la 1 până la n. Aceasta înseamnă că trebuiesc parcurse toate valorile de la 1 la n şi trebuiesc înmulţite. Pentru aceasta este potrivită instrucţiunea *for* cu limitele 1 şi n. La fiecare pas al lui *for* o valoare *i* din intervalul [1,n] va fi înmulţită la *fact*.

Variabila *fact* este iniţializată cu 1 şi la final, în urma înmulţirilor repetate va conţine valoarea factorialului lui n. Deoarece factorialul are o creştere foarte rapidă, variabila *fact* a fost declarată de tip *long* (cel mai mare interval de numere întregi din C#).

10. *Calculaţi suma cifrelor unui număr natural dat cu maximum 9 cifre.*

Raţionament

Pentru a rezolva această problemă trebuie să ne folosim de "uneltele" de care dispunem până în acest moment. Problema care se pune este cum să obţinem cifrele individuale ale numărului dat. O modalitate simplă şi usor de implementat este de a începe "ruperea" numărului în cifre începând cu cea mai din dreapta (cea mai nesemnificativă), fapt care se poate realiza calculând restul împărţirii la 10 a numărului dat. În acest fel se face primul pas. Trebuiesc însă obţinute toate cifrele numărului, în vederea însumării lor. Acest lucru îl vom realiza în acelaşi mod în care am obţinut prima cifră şi anume: după obţinerea primei cifre, vom împărţi numărul dat la 10; această operaţie va avea ca şi efect "pierderea" ultimei cifre din numărul iniţial. În continuare, pentru numărul nou obţinut calculăm restul împărţirii la 10 şi vom obţine ca şi mai sus, ultima cifră a sa, pe care o vom adăuga-o la sumă. Dacă mergem puţin înapoi vom observa că această a două cifră obţinută reprezintă de fapt penultima cifră a numărului iniţial. Vom continua acest raţionament de împărţire a numerelor şi de adăugare a ultimei cifre la suma până când la o împărţire la 10 vom obţine câtul 0. În acel moment vom şti suma tuturor cifrelor numărului iniţial.

Program

```
using System;
namespace ConsoleApplication10 {
    class Program
    {
        static void Main()
        {
            int n, m, suma = 0;
            Console.Write("Dati un număr de maxim 9 cifre : ");
            n = int.Parse(Console.ReadLine());
            m = n;
            while (m != 0)
            {
                suma += m % 10; // suma=suma+m%10;
                m /= 10; // m=m/10;
            }
            Console.WriteLine("Suma cifrelor lui {0} este {1}", n, suma);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

- pe prima linie sunt declarate variabilele m şi n de tip întreg şi variabila s, în care vom calcula

suma cifrelor lui n (observam iniţializarea acesteia de la declarare cu 0, deoarece însumarea cifrelor se va face pornind de la 0);
- citirea de la tastatură numărului n;
- lui m i se atribuie variabila n, pentru a nu pierde valoarea n; aceasta este o tehnica utilizata întotdeauna când avem de modificat valoarea reţinută de o variabila, însă nu dorim sa pierdem valoarea iniţială. Dacă nu am folosi atribuirea m=n, am împărţit variabila n la 10 până când valoarea acesteia ar deveni 0 şi evident nu am mai şti care a fost valoarea iniţială. Utilizând însa atribuirea m=n, valoarea lui n rămâne nealterata.
- urmează o instrucţiune *while* care conţine două instrucţiuni de atribuire compusă (în prima se adaugă la s ultima cifră a lui m, iar în a două se trunchiază m de ultima cifră); aceste instrucţiuni se repetă până când m va avea valoarea 0 ;
- se afişează rezultatul folosind o funcţie *Console.WriteLine()*; se vor afişa valorile reţinute de variabilele n şi s.

Efect

Introduceţi numărul: 247
Suma cifrelor lui 247 este 13.

Exemplu de funcţionare pas cu pas a programului

Sa consideram ca n=247
Urmărind linie cu linie programul de mai sus vom obţine:
s=0
Daţi un număr de maxim 9 cifre: 247
m=247
m=247 != 0
  s=s+247%10 ⇔ s=0+7=7
  m=m/10 ⇔ m=24

m=24 != 0
  s=s+24%10 ⇔ s=7+4=11
  m=m/10 ⇔ m=2
m=2 != 0
  s=s+2%10 ⇔ s=11+2=13
  m=m/10 ⇔ m=0
m=0 => s=13
Suma cifrelor lui 247 este 13.

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication10b
{
    class Program
    {
        static int suma(int nr)
        {
            int s = 0;
            while (nr != 0)
            {
                s += nr % 10;
                nr /= 10;
            }
```

```
            return s;
        }
        static void Main()
        {
            int n;
            Console.Write("Dati un număr de maxim 9 cifre : ");
            n = int.Parse(Console.ReadLine());
            Console.WriteLine("Suma cifrelor lui {0} este {1}", n, suma(n));
            Console.ReadKey();
        }
    }
}
```

11. *Să se scrie un program care să calculeze „cifra de control” a unui număr natural. Aceasta se obţine însumând cifrele numărului şi dacă suma obţinută este >=10 se repetă algoritmul până când se obţine o sumă formată dintr-o singură cifră (deci un număr <10).*
Ex: n=9989879 → s=9+9+8+9+8+7+9=59 → s=5+9=14 → **s**=1+4=**5**

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication11 {
    class Program
    {
        static int suma(int nr)
        {
            int s = 0;
            while (nr != 0)
            {
                s += nr % 10;
                nr /= 10;
            }
            return s;
        }
        static void Main()
        {
            int n, control;
            Console.Write("Dati un număr de maxim 9 cifre : ");
            n = int.Parse(Console.ReadLine());
            do
            {
                Console.Write(" -> ");
                control = suma(n);
                Console.Write(control);
                n = control;
            } while (control >= 10);
            Console.WriteLine("\nCifra de control a nr de mai sus este {0}",
control);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

– se citeşte numărul n de la tastatură

– pornind de la numărul iniţial, cu ajutorul instrucţiunii *while*, se calculează suma cifrelor numărului atâta timp cât ea este >=10; dacă este >=10 se va calcula suma cifrelor pentru acest nou număr. Calculul de opreşte în momentul în care suma obţinută este <10, iar aceasta va reprezenta "cifra de control" a numărului iniţial
– funcţia *suma* calculează suma cifrelor unui număr primit ca şi parametru şi returnează valoarea obţinută

12. *Se citeşte de la tastatură un număr natural n≤9. Să se afişeze toate numerele de n cifre care adunate cu răsturnatul lor dau un pătrat perfect.*

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication12 {
    class Program
    {
        static int rasturnat(int nr)
        {
            int r = 0;
            while (nr != 0)
            {
                r = (r*10)+nr % 10;
                nr = nr/10;
            }
            return r;
        }
        static void Main()
        {
            int sum, rsum, n, i, start, stop;
            Console.Write("Cate cifre? ");
            n = int.Parse(Console.ReadLine());
            start=(int)Math.Pow(10, n-1);
            stop=(int)Math.Pow(10, n);
            for (i = start; i < stop; i++)
            {
                sum = i + rasturnat(i);
                rsum = (int)Math.Sqrt(sum);
                if ( (rsum*rsum) == sum)
                    Console.WriteLine("{0} + {1} = {2} este patrat perfect:
{3}^2", i,rasturnat(i),sum,rsum);
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se citeşte de la tastatură numărul de cifre *n*. Conform acestuia, verificările se vor face pe intervalul de valori $[10^{n-1}, 10^n-1]$ (Ex: n=3 ⇒ [100, 999]). Pentru a ridica pe 10 la o putere se foloseşte metoda *Math.Pow*, căreia i se mai aplică suplimentar şi operatorul de cast *(int)* deoarece rezultatul returnat de metoda *Math.Pow* este de tip *double* şi nu ar fi posibilă atribuirea unei astfel de valori la o variabilă de tip *int*.

Cu ajutorul instrucţiunii *for* se parcurg toate valorile de la 10^{n-1} la 10^{n}-1. Fiecare din ele se adună cu răsturnatul lor. Răsturnatul se obţine cu ajutorul unei funcţii care este asemănătoare cu cea de obţinere a sumei cifrelor unui număr.

Pentru a testa dacă suma dintre un număr şi răsturnatul său este pătrat perfect se procedează astfel: Se calculează în variabila *rsum* radicalul sumei cu ajutorul metodei *Math.Sqrt*, căreia i se mai aplică suplimentar şi operatorul de cast *(int)* deoarece rezultatul returnat de metoda *Math.Sqrt* este de tip *double* şi nu ar fi posibilă atribuirea unei astfel de valori la o variabilă de tip *int*. Se testează apoi dacă pătratul acestei valori este egal cu suma dintre număr şi răsturnatul său (Ex 1: i=346, rasturnat(i)=643, sum=989, rsum=31, rsum*rsum=961≠989, deci nu are loc egalitatea; Ex 2: i=164, rasturnat(i)=461, sum=625, rsum=25, rsum*rsum=626=625, deci are loc egalitatea).

13. *Se citeste de la tastatură un şir de numere întregi până la citirea lui 0. Să se afişeze valoarea minimă şi maximă citită şi media lor aritmetica (fără vectori).*

Program

```
using System;
namespace ConsoleApplication13
{
    class Program
    {
        static void Main(string[] args)
        {
            int nr, min, max, tot=1;
            float sum, ma;
            Console.WriteLine("Se vor citi numere pana la intr. lui 0");
            Console.Write("Dati un număr: ");
            nr = int.Parse(Console.ReadLine());
            min = max = nr;
            sum = nr;
            do
            {
                if (nr < min)
                    min = nr;
                if (nr > max)
                    max = nr;
                Console.Write("Dati un număr: ");
                nr = int.Parse(Console.ReadLine());
                sum = sum + nr;
                tot++;
            } while (nr != 0);
            tot--;
            ma = sum / tot;
            Console.Write("min={0}, max={1}, media aritm.={2}", min, max, ma);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

În prima fază se citeşte de la tastatură primul număr şi se iniţializează minimul, maximul şi suma numerelor cu această valoare. Pentru ca la final să se poată calcula media aritmetică a numerelor introduse, se foloseşte variabile *tot* care iniţial are valoarea 1 şi care se va incrementa cu 1 la citirea fiecărui nou număr.

În cadrul unei instrucţiuni repetitive *do-while* se vor citi numere de la tastatură până la introducerea lui 0. Cu fiecare număr nou citit se fac câteva operaţii: se compară cu minimul curent şi dacă este mai mic decât acesta se actualizează minimul, se compară cu maximul curent şi dacă este mai mare decât acesta se actualizează maximul, se adună la suma numerelor.

După ce s-a citit de la tastatură valoarea 0, se decrementează cu 1 valoarea variabilei *tot* pentru a exclude ultima valoare citită care era 0 şi care nu mai ia parte la calcule, după care se calculează media aritmetică, împărţind suma numerelor citite la totalul lor. Se observă că variabila *sum* a fost declarată de tip *float* deşi ea conţine suma unor numere întregi. S-a apelat la acest artificiu deoarece dacă era declarată de tip *int* şi la final efectuam operaţia sum / tot, aceasta implicând numere întregi, dădea tot un număr întreg (Ex: 27/5=5, deşi media aritmetică era 5.4). Folosind însă variabila *sum* de tip *float*, rezultatul operaţiei sum / tot va fi şi el de tip *float*, adică exact ceea ce avem nevoie.

La final se afişează valoare minimă, maximă şi media aritmetică.

Se sublinia în enunţ ca rezolvarea să nu implice vectori, deoarece se observă că ei nu sunt necesari pentru a memora toate valorile introduse de la tastatură. Este suficientă o variabilă simplă cu ajutorul căreia se va citi câte o valoare, se fac operaţiile necesare, după care poate stoca o nouă valoare, cea veche nemaifiind necesară.

14. *Fiind date două numere naturale n şi m, să se formeze un nou număr care să conţină cifrele maxime de pe fiecare poziţie din n şi m.* Ex: n=2618, m=3456 → 3658

Program

```
using System;
namespace ConsoleApplication14
{
    class Program
    {
        static void Main()
        {
            int n, m, max = 0, p10 = 1, cifra;
            Console.Write("Introduceti primul număr: ");
            n = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            m = int.Parse(Console.ReadLine());
            while (n != 0 || m != 0)
            {
                if (n % 10 > m % 10)
                    cifra = n % 10;
                else
                    cifra = m % 10;
                max = max + cifra * p10;
                p10 *= 10;
                n /= 10;
                m /= 10;
            }
            Console.WriteLine("Noul număr este : {0}", max);
            Console.ReadKey();
        }
    }
}
```

Observaţie :

```
if ((n%10)>(m%10))
       cifra = n%10;
else                           ⇔  cifra=(n%10)>(m%10) ? n%10 : m%10;
       cifra = m%10;
```

Analiza programului

Această problemă derivă şi ea din problema aflării cifrelor unui număr natural. Rezolvarea acestei probleme merge pe ideea de a descompune în paralel cele două numere n şi m, până când ambele devin 0 (e posibil ca acest lucru să se întâmple în momente diferite dacă cele două numere nu au acelaşi număr de cifre).

Cifrele de pe aceeaşi poziţie obţinute din cele două numere se compară şi care este mai mare va face parte din numărul care se doreşte să se obţină.

Noul număr *max* se obţine printr-un procedeu invers celui de obţinere a cifrelor unui număr, în sensul că de data aceasta câte o cifră maximă obţinută de la cele două numere iniţiale se înmulţeşte cu 10 la o anumită putere corespunzătoare poziţiei cifrei în numărul *max* (prima cifră - cea mai din dreapta - se înmulţeşte cu 10^0, a două cifră cu 10^1, a treia cu 10^2, ş.a.m.d.) şi se însumează. Înmulţirea cu 10 la o putere corespunzătoare ajută a plasa o cifră în poziţia care trebuie în numărul care se doreşte să se obţină *max*.

Se observă că nu s-a folosit metoda *Math.Pow*, ci s-a aplicat o metoda optimizată din punct de vedere al calculelor, folosind variabila *p10* în construirea valorii 10^{putere}. Variabila *p10* are la început valoarea 1 şi pe parcurs, prin înmulţire cu 10 va avea valoarea 10^1, 10^2, etc. Aceasta este o variantă mai bună decât a folosi metoda *Math.Pow*, care pentru a calcula 10^{putere} foloseşte mai mult de o înmulţire, pe când varianta utilizată în acest program foloseşte numai una, având deja calculat de la pasul anterior $10^{putere-1}$.

Exemplu : n=2618, m=3456

(8,6) → $8*10^0$+
(1,5) → $5*10^1$+ → 8+50+600+3000=3658
(6,4) → $6*10^2$+
(2,3)→ $3*10^3$

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication14b
{
    class Program
    {
        static int cifre_maxime(int n1, int n2)
        {
            int n3 = 0, cifra, p10 = 1;
            while (n1 != 0 || n2 != 0)
            {
                if (n1 % 10 > n2 % 10)
                    cifra = n1 % 10;
                else
                    cifra = n2 % 10;
                n3 += cifra * p10;
                p10 *= 10;
                n1 /= 10;
                n2 /= 10;
            }
            return n3;
        }
```

```
        static void Main()
        {
            int n1, n2;
            Console.Write("Introduceti primul număr: ");
            n1 = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            n2 = int.Parse(Console.ReadLine());
            Console.WriteLine("Numărul format este {0}", cifre_maxime(n1, n2));
            Console.ReadKey();
        }
    }
}
```

15. *Să se determine c.m.m.d.c. a două numere naturale. (2 variante)*

Varianta 1:

Program

```
using System;

namespace ConsoleApplication15
{
    class Program
    {
        static void Main()
        {
            int nr1, nr2, n, m;
            Console.Write("Introduceti primul număr: ");
            nr1 = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            nr2 = int.Parse(Console.ReadLine());
            n = nr1; m = nr2;
            while (nr1 != nr2)
                if (nr1 > nr2)
                    nr1 -= nr2;
                else
                    nr2 -= nr1;
            Console.WriteLine("C.m.m.d.c.({0},{1}) = {2}", n, m, nr1);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Aceasta este o metodă foarte simplu de aplicat şi de reţinut în aflarea c.m.m.d.c. a două numere, însă probabil necesită mai multe calcule decât alţi algoritmi.

Ideea acestui algoritm este următoarea: atâta timp cât cele două numere date sunt diferite, din cel mai mare se scade cel mai mic, iar prin acest procedeu la un moment dat cele două numere vor deveni egale. Acea valoare finală pe care o au ambele numere reprezintă c.m.m.d.c. al lor.

Având în vedere faptul că ambele numere vor fi alterate în acest calcul, pentru a nu pierde valoarea lor iniţială, înainte de instrucţiunea *while* se poate face câte o copie a lor cu care să se lucreze mai departe.

Varianta 2:

Algoritmul lui Euclid

Program

```
using System;
namespace ConsoleApplication15b {
    class Program {
        static void Main()
        {
            int nr1, nr2, temp, r, n, m;
            Console.Write("Introduceti primul număr: ");
            nr1 = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            nr2 = int.Parse(Console.ReadLine());
            n = nr1; m = nr2;
            if (nr1 < nr2)
            {
                temp = nr1;
                nr1 = nr2;
                nr1 = temp;
            }
            do
            {
                r = nr1 % nr2;
                nr1 = nr2;
                nr2 = r;
            } while (r != 0);
            Console.WriteLine("C.m.m.d.c.({0},{1}) = {2}", n, m, nr1);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Acesta este un algoritm destul de celebru de determinare a c.m.m.d.c a două numere naturale. Ideea pe care se bazează însă este puţin mai dificilă decât cea a primului algoritm prezentat, însă cel mai probabil că necesită mai puţin calcul.

Prima dată se citesc cele două numere şi se asigură că primul dintre ele este cel mai mare.

Ideea algoritmului lui Euclid este de a începe prin a împărţi numărul mai mare din perechea de numere date, la cel mai mic. Dacă restul obţinut este diferit de 0, se repetă împărţirea, de data aceasta împărţindu-se împărţitorul la restul de la împărţirea anterioară. Acest mecanism se repetă până când se obţine restul 0. În acel moment este aflat c.m.m.d.c., iar el este penultimul rest obţinut (nenul).

În cazul în care se obţine din start restul 0 înseamnă că numărul mai mic din cele două reprezintă c.m.m.d.c.

Ex : (60, 25)=5

60 : 25=2 rest 15

25 : 15=1 rest 10

15 : 10=1 rest **5**

10 : 5=2 rest 0

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication15c {
```

```
    class Program {
        static int cmmdc(int n1, int n2)
        {
            int r;
            do
            {
                r = n1 % n2;
                n1 = n2;
                n2 = r;
            } while (r != 0);
            return n1;
        }
        static void Main()
        {
            int a, b, temp;
            Console.Write("Introduceti primul număr: ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            b = int.Parse(Console.ReadLine());
            if (a < b)
            {
                temp = a;
                a = b;
                b = temp;
            }
            Console.WriteLine("C.m.m.d.c.({0},{1}) = {2}", a, b, cmmdc(a, b));
            Console.ReadKey();
        }
    }
}
```

16. *Să se afişeze toţi divizorii (proprii) comuni a două numere naturale.*

Program

```
using System;
namespace ConsoleApplication16
{
    class Program
    {
        static void Main()
        {
            int m, n, c;
            Console.Write("Introduceti primul număr: ");
            m = int.Parse(Console.ReadLine());
            Console.Write("Introduceti al doilea număr: ");
            n = int.Parse(Console.ReadLine());
            if (m >= n) c = n / 2;
            else c = m / 2;
            Console.WriteLine("Divizorii comuni pentru {0} şi {1}", m, n);
            for (int i = 2; i <= c; i++)
                if ((n % i == 0) && (m % i == 0))
                    Console.Write(i + " ");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Când se pune problema determinării divizorilor unui număr n, la început se pune întrebarea: în ce mulţime de valori de va face căutarea divizorilor? Un răspuns ar putea fi intervalul [1, n]. În cazul în care nu dorim şi divizorii impropri (1 şi numărul însuşi) ar rămâne intervalul [2, n-1]. În situaţia aceasta însă ar trebui să ne punem întrebarea: care este cel mai mare divizor propriu al unui număr n? Fără prea multe dificultăţi ar trebui să realizăm că această valoare este la jumătatea numărului, deoarece aceasta înmulţită cu 2 ar da numărul n. De la jumătate încolo, orice valoare am înmulţi cu 2, va trece de valoarea lui n, deci nu mai are cum să fie divizor al lui n. Astfel, deducem că divizorii proprii îi vom căuta în intervalul [2, n/2].

În problema de faţă, având de-a face cu divizorii a două numere, din nou trebuie luată o decizie asupra intervalului în care se vor căuta divizorii comuni. Ar putea apărea mai multe variante de răspuns, însă cea corectă este de a căuta în intervalul de la 2 până la jumătatea numărului mai mic din cele două. De ce până la jumătatea numărului mai mic? Deoarece dacă de exemplu s-ar merge până la jumătatea numărului mai mare, valorile de la jumătatea numărului mai mic până la jumătatea celui mai mare sigur nu vor mai putea fi divizori ai numărului mai mic din considerentele enunţate în paragraful anterior.

Divizor comun înseamnă o valoare la care se împart exact ambele numere. De fiecare dată când va fi găsită o astfel de valoare, ea va fi afişată pe ecran.

Ex : n=60 → intervalul de căutare [2, 30]
m=42→ intervalul de căutare [2, 21]

Dacă s-ar mai căuta divizori comuni în intervalul [22, 30] nu s-ar mai găsi nici unul, deoarece orice valoare >=22 nu mai poate fi divizor propriu al lui 42. Din acest motiv, divizorii comuni ai celor două numere se caută până la jumătatea celui mai mic dintre ele.

Pentru a decide care dintre numerele n şi m este mai mare s-a utilizat instrucţiunea

```
c = m>=n ? n/2 : m/2;
```

Aceasta este echivalentă cu o instrucţiune *if-else*:

```
if (m>=n)
      c = n/2;
else
      c = m/2;
```

S-a optat pentru operatorul de decizie ?: mai mult din motive de economie de scriere decât din alte considerente. Astfel, dacă m>=n rezultatul evaluării va fi n/2 care apoi i se atribuie lui c, altfel va fi m/2 care se va atribui lui c.

17. *Să se determine toate numerele "perfecte" mai mici decât 10000. Un număr este perfect dacă este egal cu suma tuturor divizorilor săi (inclusiv 1).*

Program

```
using System;

namespace ConsoleApplication17
{
    class Program
    {
        static void Main()
        {
            int nr, i, s;
            Console.WriteLine("Nr perfecte < 10000 sunt:");
            for (nr = 4; nr <= 10000; nr++)
            {
                s = 0;
```

```
                for (i = 1; i <= nr / 2; i++)
                    if (nr % i == 0)
                        s += i;
                if (s == nr)
                    Console.WriteLine(nr);
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Acest program necesită aflarea tuturor divizorilor unui număr (inclusiv 1) şi însumarea lor. Cu ajutorul unei instrucţiuni *for* se vor testa toate numere naturale de la 4 (cel mai mic număr natural care are divizori proprii) până la 10000. S-a discutat în problema anterioară modul de aflare a tuturor divizorilor unui număr natural. În cazul în care suma divizorilor este egală cu numărul analizat, acesta va fi tipărit pe ecran.

<u>Cu funcţie</u> :

Program

```
using System;

namespace ConsoleApplication17b
{
    class Program
    {
        static int suma_div(int nr)
        {
            int s = 1, i;
            for (i = 2; i <= nr / 2; i++)
                if (nr % i == 0)
                   s += i;
            return s;
        }

        static void Main()
        {
            int n, i;
            Console.Write("Introduceti numărul n= ");
            n = int.Parse(Console.ReadLine());
            Console.WriteLine("Numerele perfecte mai mici decat {0} sunt", n);
            for (i = 4; i <= n; i++)
                if (suma_div(i) == i)
                    Console.WriteLine(i);
            Console.ReadKey();
        }
    }
}
```

18. *Testaţi dacă un număr natural dat este prim. (Prin număr prim înţelegem orice număr natural care se împarte doar la 1 şi la el însuşi; se considera ca 2 este cel mai mic număr prim).*

Program

```
using System;
namespace ConsoleApplication18 {
```

```
    class Program
    {
        static void Main()
        {
            int i, n, rad_n;
            bool prim = true;
            Console.Write("Introduceti n = ");
            n = int.Parse(Console.ReadLine());
            rad_n = (int)Math.Sqrt(n);
            for (i = 2; i <= rad_n; i++)  //for (i=2; i*i<=n; i++)
                if (n % i == 0)
                {
                    prim = false;
                    break;
                }
            if (prim)
                Console.WriteLine("Numărul {0} este prim", n);
            else
                Console.WriteLine("Numărul {0} nu este prim", n);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

- în scopul verificării dacă un anumit număr este prim, va trebui sa testam dacă se împarte la vreun alt număr înafara de 1 şi el însuşi. Problema care se pune este în ce interval vom căuta posibilele valori la care s-ar putea împărţi numărul. S-ar putea propune variantele de intervale [2, n-1] sau [2, n/2], însa varianta optima este de a căuta în intervalul [2, $\sqrt{n}$]. Astfel, se vor parcurge cu ajutorul unei instrucţiuni *for* toate numerele naturale cuprinse între 2 şi $\sqrt{n}$; dacă n se divide cu vreunul dintre ele (restul împărţirii lui n la i este 0), atunci se opreşte forţat instrucţiunea *for* cu ajutorul instrucţiunii *break* şi variabila *prim* ia valoarea *false*. Dacă s-a terminat normal instrucţiunea *for* înseamnă că numărul *n* este prim, deoarece nu s-a găsit nici un divizor, iar variabila *prim* are valoarea *true*;
- se observă că înainte de instrucţiunea *for*, am depus valoarea lui $\sqrt{n}$ în variabila *rad_n*. Scopul acestei atribuiri este de a evita calculul radicalului la fiecare pas de *for*. De reţinut că această operaţie este util de aplicat oricând apare un calcul care nu se modifică (constant), într-o instrucţiune repetitivă. Astfel, el poate fi efectuat o singură dată, înainte de a intra în instrucţiunea repetitivă.
- cu ajutorul unei instrucţiuni *if* testăm valoarea variabilei *prim*, astfel dacă ea este *true* se va afişa mesajul ca *n* este număr prim şi altfel se va afişa mesajul ca *n* nu este număr prim.

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication18b {
    class Program
    {
        static bool test_prim(int n)
        {
            int i, rad_n;
            bool prim = true;
            rad_n = (int)Math.Sqrt(n);
```

```
            for (i = 2; i <= rad_n; i++)  //for (i=2; i*i<=n; i++)
                if (n % i == 0)
                {
                    prim = false;
                    break;
                }
            return prim;
        }
        static void Main()
        {
            int n;
            Console.Write("Introduceti n = ");
            n = int.Parse(Console.ReadLine());
            if (test_prim(n))
                Console.WriteLine("Numărul {0} este prim", n);
            else
                Console.WriteLine("Numărul {0} nu este prim", n);
            Console.ReadKey();
        }
    }
}
```

19. *Se citeşte de la tastatură un număr natural x mai mare decât 2. Să se găsească p şi q numere prime astfel încât p<x<q, iar diferenţa q-p este minimă.*

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication19 {
    class Program {
        static bool test_prim(int n)
        {
            int i, rad_n;
            bool prim = true;
            rad_n = (int)Math.Sqrt(n);
            for (i = 2; i <= rad_n; i++)  //for (i=2; i*i<=n; i++)
                if (n % i == 0)
                {
                    prim = false;
                    break;
                }
            return prim;
        }
        static void Main()
        {
            int x, p, q;
            bool prim;
            Console.Write("Introduceti x = ");
            x = int.Parse(Console.ReadLine());
            p = x;
            do
            {
                p--;
                prim = test_prim(p);
            } while (prim == false);
            q = x;
            do {
```

```
                q++;
                prim = test_prim(q);
            } while (prim == false);
            Console.Write("Soluţia găsita:{0}<{1}<{2} ", p, x, q);
            Console.Write("iar {0}-{1}={2}", q, p, q - p);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Aceasta problemă presupune de fapt găsirea primului număr prim mai mic decât *x* şi a primului număr prim mai mare decât *x*. În felul acesta diferenţa *q-p* va fi minimă.

În acest sens, se porneşte prima dată de la valoarea *x-1* şi se testează în jos cu ajutorul unei instrucţiuni repetitive *do-while* toate valorile până când se va găsi primul număr prim. Acela va fi numărul *p*. După aceea, se porneşte de la valoarea *x+1* tot cu o instrucţiune repetitivă *do-while* şi se testează în sus toate valorile până când se va găsi primul număr prim. Acela va fi numărul *q*.

Modul de testare a unui număr dacă este prim este acelaşi cu cel de la problema anterioară.

20. *Se citeşte de la tastatură un număr natural par. Să se decidă dacă acesta poate fi scris ca şi suma de două numere prime şi să se afişeze toate soluţiile găsite (se va considera că şi 1 este număr prim).* (Conjectura lui Goldbach: "Orice număr par mai mare decât 2 este suma a două numere prime.")

Cu funcţie :

Program

```
using System;
namespace ConsoleApplication20 {
    class Program {
        static bool test_prim(int n)
        {
            int i, rad_n;
            bool prim = true;
            rad_n = (int)Math.Sqrt(n);
            for (i = 2; i <= rad_n; i++)  //for (i=2; i*i<=n; i++)
                if (n % i == 0)
                {
                    prim = false;
                    break;
                }
            return prim;
        }
        static void Main()
        {
            int nr, nr1, nr2 = 0;
            bool prim;
            Console.Write("Introduceti nr: ");
            nr = int.Parse(Console.ReadLine());
            for (nr1 = 1; nr1 <= nr / 2; nr1 = nr1 + 2)
            {
                prim = test_prim(nr1);
                if (prim == true)
                {
                    nr2 = nr - nr1;
```

```
                prim = test_prim(nr2);
            }
            if (prim == true) //nr=nr1+nr2 şi nr1, nr2 sunt prime
                Console.WriteLine("Solutie:{0}+{1}",nr1,nr2);
        }
        Console.ReadKey();
    }
  }
}
```

Analiza programului

În cadrul acestei probleme, în încercarea de a scrie numărul *nr* ca şi sumă de două numere prime se vor căuta valori în intervalul [1, nr/2]. Dacă se găseşte un număr prim *nr1* din acest interval, se va testa apoi dacă şi *nr2=nr-nr1* este şi el prim. Dacă da, aceasta înseamnă ca nr=n1+nr2, iar *nr1* şi *nr2* sunt prime. Această pereche se va tipări pe ecran. *nr1* se caută în intervalul [1, nr/2] deoarece dacă se trece de jumătatea numărului, se vor obţine perechi duplicate. De asemenea, valorile lui *nr1* în cadrul instrucţiunii *for* merg din 2 în 2, deoarece se merge doar pe valori impare.

Ex : Dacă nr=26, atunci s-ar obţine soluţia nr1=3, nr2=23 dar şi nr1=23, nr2=3.

Modul de testare a unui număr dacă este prim este acelaşi cu cel de la problema 18.

21. *Se citeşte de la tastatură un număr natural. Să se decidă dacă acesta poate fi scris ca şi sumă de două patrate şi să se afişeze toate soluţiile găsite.*

Program

```
using System;

namespace ConsoleApplication21 {
    class Program
    {
        static void Main()
        {
            int n, i, j, t1, t2, rad_n, sol=0;
            Console.Write("Introduceti numărul n=");
            n = int.Parse(Console.ReadLine());
            rad_n = (int)Math.Sqrt(n);
            for (i = 1; i <= rad_n; i++)
            {
                t1 = i * i;
                j = (int)Math.Sqrt(n - i * i);
                t2 = j * j;
                if ((t1 <= t2) && (n == t1 + t2))
                {
                    Console.WriteLine("Solutie: {0}={1}*{2}+{3}*{4}", n, i, i,
j, j);
                    sol++;
                }
            }
            Console.Write("Au fost gasite {0} solutii", sol);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Pentru un *n* citit de la tastatură se testează toate valorile din intervalul $[1, \sqrt{n}]$ pentru a vedea dacă poate fi îndeplinită condiţia din enunţ. Astfel, pentru fiecare valoare *i* din acest interval se calculează $j=\sqrt{n-i*i}$. În cazul în care n=i*i+j*j înseamnă că a fost găsită soluţia (*i, j)*. Intervalul de testare este $[1, \sqrt{n}]$ deoarece dacă $i=\sqrt{n}$, atunci i*i=n, deci nu se mai poate găsi un al doilea număr pozitiv *j* pentru care i*i+j*j=n.

Pentru a ne verifica, (*i, j)* este soluţie a problemei deoarece **n**=i*i+j*j= $i*i+\sqrt{n-i*i}*\sqrt{n-i*i}$=i*i+n-i*i=**n**. S-a mai adăugat şi testul dacă t1<=t2 pentru a evita generarea de soluţii identice. Ex: (2, 5), (5, 2)

La final se afişează numărul total de soluţii gasite (care poate fi şi 0).

<u>Exemplu</u> : n=29, $i\in[1, 5]$

i=1, i*i=1, n-i*i=28, $j=\sqrt{28}$=5, i*i+j*j=**26≠28**

i=2, i*i=4, n-i*i=25, $j=\sqrt{25}$=5, i*i+j*j=**28=28** → (2, 5)

...

22. *Să se afişeze primele n perechi de numere prime care sunt consecutive în mulţimea numerelor impare.*

<u>Cu funcţie</u> :

Program

```
using System;
namespace ConsoleApplication22 {
    class Program {
        static bool test_prim(int n)
        {
            int i, rad_n;
            bool prim = true;
            rad_n = (int)Math.Sqrt(n);
            for (i = 2; i <= rad_n; i++)  //for (i=2; i*i<=n; i++)
                if (n % i == 0)
                {
                    prim = false;
                    break;
                }
            return prim;
        }
        static void Main()
        {
            int n, i, c;
            Console.Write("Introduceti numărul de perechi n=");
            n = int.Parse(Console.ReadLine());
            c = 0;  i = 3;
            while (c < n)
            {
                if (test_prim(i) && test_prim(i + 2))
                {    // i, i+2 sunt prime
                    Console.WriteLine("{0}, {1}", i, i + 2);
                    c++;
                }
                i = i + 2;
            } //while
            Console.ReadKey();
```

```
        }
    }
}
```

Analiza programului

Soluţiile acestei probleme se vor căuta începând cu numărul 3, după care se va merge din 2 în 2. Astfel se vor testa perechile de valori (3, 5), (5, 7), (7, 9), ş.a.m.d. Prima dată se testează prima valoare a perechii. Dacă este număr prim se trece la testarea celei de-a doua valori (egală cu prima valoare + 2). Dacă şi ea este număr prim înseamnă că a fost găsită o soluţie, care se tipăreşte pe ecran. De fiecare dată când se găseşte o soluţie, se va incrementa un contor *c*, pentru a se şti când vor fi găsite cele *n* perechi căutate. Mecanismul de testare dacă un număr este prim este acelaşi cu cel de la problema 21.

Optimizare :

```
while (c < n)
{
    if (test_prim(i))
        if (test_prim(i + 2))
        {   // i, i+2 sunt prime
            //Console.WriteLine("{0}, {1}", i, i + 2);
            c++;
            i = i + 2;
        }
        else i = i + 4;
    else i = i + 2;
}
```

Dacă în perechea (*i*, *i+2*), *i+2* nu este număr prim, atunci nu se va mai testa perechea (*i+2*, *i+4*), ci se va trece direct la testarea perechii (*i+4*, *i+6*). Ex: (7, 9): i=7 este nr prim, i+2=9 nu este nr prim ⇒ nu se va mai testa perechea (9, 11), ci se va trece la testarea perechii (11, 13). În felul acesta am economisit de la testare o pereche de valori şi implicit am câştigat puţin timp la rularea programului, ceea ce pentru un *n* suficient de mare poate să devină un timp simţitor.

23. *Să se descompună un număr natural n în factori primi.* Ex: $360=2^3*3^2*5^1$

Program

```
using System;
namespace ConsoleApplication23
{
    class Program
    {
        static void Main()
        {
            int n, m, i, putere;
            Console.Write("Introduceti numărul n = ");
            n = int.Parse(Console.ReadLine());
            Console.Write("{0} = ", n);
            m = n;
            for (i = 2; i <= n / 2; i++)
            {
                if (m % i == 0)
                {
                    putere = 0;
                    while (m % i == 0)
                    {
                        putere++;
```

```
                    m = m / i;
                }
                Console.Write("{0}^{1} * ", i, putere);
            }
            if (m == 1)
                break;
        }
        Console.Write("1");
        Console.ReadKey();
    }
  }
}
```

Analiza programului

Această problemă îşi propune să afle toţi divizorii primi ai unui număr natural *n* şi puterea la care intră acei divizori în descompunerea lui *n*. În acest sens, se vor căuta toţi divizorii lui *n* în intervalul [2, n/2]. De fiecare dată când este găsit un divizor, se va calcula puterea la care intră acel divizor în descompunerea lui *n*, împărţind în mod repetat acel divizor la *n* până când acest lucru nu mai este posibil. Noul număr *n* va fi *n*-ul iniţial împărţit la divizorul găsit la puterea maximă ce intră în descompunerea lui *n*, după care se va trece să se testeze o nouă valoare. Calculul se va opri în momentul în care *n* devine 1.

Ex : n=360

Primul divizor găsit este 2. Se împarte *n* la 2 până când acest lucru nu mai este posibil: 360:2=180:2=90:2=45. Noul *n* cu care se va lucra mai departe va fi 45, şi s-a aflat că 2^3 intră în descompunerea *n*-ului iniţial.

Se va trece la testarea lui 3. Deoarece 3 este divizor al lui 45 se trece la aflarea puterii la care intră 3 în descompunerea lui *n*: 45:3=15:3=5. Noul *n* cu care se va lucra mai departe va fi 5, şi s-a aflat că 3^2 intră în descompunerea *n*-ului iniţial.

Se va trece la testarea lui 4, dar acesta nu este divizor al lui 5.

Se va trece la testarea lui 5. Deoarece 5 este divizor al lui 5 se trece la aflarea puterii la care intră 5 în descompunerea lui *n*: 5:5=1. S-a aflat că 5^1 intră în descompunerea *n*-ului iniţial. Acum *n* a devenit 1, moment în care calculul se opreşte.

Din rezolvarea acestei probleme se observă că atunci când este găsit un divizor, acesta nu este testat dacă este prim sau nu. De ce? Deoarece nu se lucrează tot timpul cu *n* iniţial, ci de la o testare a unui divizor la alta, *n* a fost împărţit la divizorii săi găsiţi până la un moment dat.

Astfel, în exemplul de mai sus, 360 a fost împărţit la 2^3, iar pe mai departe nu s-a mai lucrat cu 360, ci cu $360:2^3$=45. Acest fapt influenţează semnificativ mersul rezolvării, deoarece numărul iniţial fiind împărţit la puterea maximă posibilă a lui 2, pe mai departe nici un multiplu al lui 2 nu va mai fi găsit ca şi divizor al unui nou *n*. Pe când dacă s-ar fi lucrat tot cu *n* iniţial, atât 4 cât şi 8 ar fi fost găsiţi ca şi divizori ai lui 360, caz în care era necesar şi testul de primalitate. În modul în care este gândită această rezolvare, orice număr se va testa şi este găsit ca şi divizor, sigur este prim, deoarece toţi posibilii săi divizori au fost deja testaţi, iar numărul iniţial a fost împărţit deja la ei.

Revenind la numărul 360, divizorii săi sunt: 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 120, 180.

În momentul în care 2 a fost găsit ca divizor al lui 360, iar apoi 360 a fost împărţit la 2^3 şi s-a obţinut 45, toţi multiplii lui 2 cad de la analizare, adică rămân de testat doar divizorii 3, 5, 9, 15, 45.

În momentul în care 3 a fost găsit ca divizor al lui 45, iar apoi 45 a fost împărţit la 3^2 şi s-a obţinut 5, toţi multiplii lui 3 cad de la analizare, adică rămân de testat doar divizorii 5, 15, 45.
În momentul în care 5 a fost găsit ca divizor al lui 5, iar apoi 5 a fost împărţit la 5^1 şi s-a obţinut 1, calculul se încheie, iar concluzia este că $360=2^3*3^2*5^1$.

24. *Se citesc de la tastatură n numere naturale şi un număr prim p. Se cere să se găsească un număr k maxim astfel încât produsul celor n numere să se dividă cu p^k, fără a calcula produsul.*
Ex: Dacă p=2, iar cele 4 numere sunt: $10=2^1*5$, $8=2^3$, $3=2^0*3$, $28=2^2*7$ → k=6

Program

```
using System;
namespace ConsoleApplication24 {
    class Program
    {
        static void Main()
        {
            int n, nr, p, k = 0, i;
            Console.Write("Dati numărul total de numere n=");
            n = int.Parse(Console.ReadLine());
            Console.Write("Introduceti numărul prim p=");
            p = int.Parse(Console.ReadLine());
            for (i = 1; i <= n; i++)
            {
                Console.Write("Introduceti un nr: ");
                nr = int.Parse(Console.ReadLine());
                while (nr % p == 0)
                {
                    k++;
                    nr = nr / p;
                }
            }
            Console.Write("k maxim={0}", k);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Rezolvarea acestei probleme merge pe ideea că puterea la care se divide *p* în cadrul produsului celor *n* numere este aceeaşi cu puterile lui *p* însumate de la cele *n* numere pe rând.

În exemplul dat 10*8*3*28=6720, iar această valoare se divide cu 2^6. Se poate observa însă şi fără a face produsul că $10*8*3*28=\underline{2^1*5}*\underline{2^3}*\underline{2^0*3}*\underline{2^2*7}=2^6*$…

Este de fapt de evitat pe cât posibil calcularea produsului deoarece la un moment dat s-ar putea obţine o valoare care să depăşească cea mai mare valoare întreagă predefinită în C# (2^{64}) . Dacă însă nu se realizează produsul, această problemă este eliminată.

Rezolvarea acestei probleme constă în a citi pe rând cele *n* numere şi a verifica puterea lui *p* care intră în descompunerea fiecăruia dintre ele. Aceste puteri se însumează, iar rezultatul obţinut este exact acel *k* maxim care se caută.

25. *Se citesc n numere naturale de la tastatură. Să se determine în câte zerouri se va termina produsul acestora, fără a calcula efectiv produsul.* Ex: 12, 35, 30, 75 → 3 zerouri

Program

```
using System;
namespace ConsoleApplication25
{
    class Program
    {
        static void Main()
        {
            int n, nr, p2 = 0, p5 = 0, i, zero;
            Console.Write("Introduceti numărul de elemente n=");
            n = int.Parse(Console.ReadLine());
            for (i = 1; i <= n; i++)
            {
                Console.Write("Introduceti un număr nr=");
                nr = int.Parse(Console.ReadLine());
                if (nr % 2 == 0)
                    while (nr % 2 == 0)
                    {
                        p2++;
                        nr /= 2;
                    }
                if (nr % 5 == 0)
                    while (nr % 5 == 0)
                    {
                        p5++;
                        nr /= 5;
                    }
            }
            if (p2 < p5)
                  zero = p2;
            else
                  zero = p5;
            Console.WriteLine("Produsul  celor  {0}  numere  se  termina  în  {1}
zerouri", n, zero);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Rezolvarea acestei probleme seamănă cu cea de la problema 24.Se merge pe aceeaşi idee că nu este necesar (şi nici recomandat) de a calcula produsul tuturor celor *n* numere pentru a afla rezultatul căutat. De data aceasta însă, pentru fiecare număr din cele *n* se va studia cu 2 la ce putere se divide şi cu 5 la ce putere se divide, deoarece fiecare 0 în care se termină produsul numerelor înseamnă o putere a lui 10, iar 10=2*5.

Toate puterile lui 2 şi 5 care apar în descompunerea celor *n* numere se însumează, iar la final care dintre cele două valori este mai mică reprezintă numărul de zerouri în care se termină produsul celor *n* numere.

Ex:12*35*30*75=$\underline{2^2*3}*\underline{5^1*7}*\underline{2^1*5^1*3}*\underline{5^2*3}=2^3*5^4*\ldots=2^3*5^3*\ldots=10^3*\ldots=1000*\ldots$ Se deduce astfel că produsul celor 4 numere se termină în 3 zerouri. (în descompunere se obţinuse 2^3 şi 5^4, 3<4 → 10^3, deci 3 zerouri).

Cu funcţie :

Program

```
using System;

namespace ConsoleApplication25b
{
    class Program
    {
        static int putere(int p, int x)
        {
            int putere_p = 0;
            while (x % p == 0)
            {
                x /= p;
                putere_p++;
            }
            return putere_p;
        }
        static void Main()
        {
            int x, p2=0, p5=0;
            Console.Write ("Introduceti un număr: ");
            x=int.Parse(Console.ReadLine());

            while (x!=0)
            {
                p2+=putere(2,x);
                p5+=putere(5,x);
                Console.Write("Introduceti un număr: ");
                x = int.Parse(Console.ReadLine());
            }
            Console.WriteLine ("Produsul se divide cu 2 la puterea {0}", p2);
            Console.WriteLine("Produsul se divide cu 5 la puterea {0}", p5);
            if (p2 >= p5) Console.WriteLine("Produsul se divide cu 10 la puterea
{0}", p5);
            else Console.WriteLine("Produsul se divide cu 10 la puterea {0}",
p2);
            Console.ReadKey();
        }
    }
}
```

26. *Sa se calculeze* n^m *efectuând mai puţin de m-1 înmulţiri. (Atenţie la valorile n şi m, ca rezultatul să nu depăşească cel mai mare întreg definit în C#)*

Program

```
using System;
namespace ConsoleApplication26 {
    class Program
    {
        static void Main()
        {
            long prod;
```

```
            int n, m, p, i, op=0;
            Console.Write("Introduceti n=");
            n = int.Parse(Console.ReadLine());
            Console.Write("Introduceti m=");
            m = int.Parse(Console.ReadLine());
            prod = n;
            p = 1;
            while ((p * 2) <= m)
            {
                prod *= prod;
                p *= 2;
                op++;
            }
            if (p < m)
                for (i = p + 1; i <= m; i++)
                {
                    prod *= n;
                    op++;
                }
            Console.WriteLine("{0}^{1}={2} din {3} operatii", n, m, prod, op);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Rezolvarea acestei probleme merge pe ideea de a ridica la putere în următorul mod: $n*n=n^2$, $n^2*n^2=n^4$, $n^4*n^4=n^8$,etc. atâta timp cât acest lucru este posibil. Când nu mai este posibil se va continua înmulţirea simplă cu câte un *n*. Ex: m=20 → n^{20} se calculează astfel: $n*n=n^2$, $n^2*n^2=n^4$, $n^4*n^4=n^8$, $n^8*n^8=n^{16}$; nu se mai poate continua în acest mod, deci până la 20 se va înmulţi cu câte un *n*: $n^{16}*n*n*n*n=n^{20}$. Se observă că faţă de a înmulţi *n* de 20 de ori, s-au făcut mai puţine înmulţiri (s-au făcut doar 7 înmulţiri).

6.2 VECTORI

27. *Dat fiind un tablou unidimensional (vector) cu numere întregi, determinaţi minimul şi maximul din acest tablou.*

Program

```
using System;

namespace ConsoleApplication27
{
    class Program
    {
        static void Main()
        {
            int n, i;
            int MIN, MAX;
            Console.Write("Dati dimensiunea vectorului A: n=");
            n = int.Parse(Console.ReadLine());
            int[] a = new int[n];
            Console.WriteLine("Dati elementele vectorului: ");
            for (i = 0; i < n; i++)
            {
                Console.Write("A[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            MIN = MAX = a[0];
            for (i = 1; i < n; i++)
            {
                if (a[i] < MIN)
                    MIN = a[i];
                if (a[i] > MAX)
                    MAX = a[i];
            }
            Console.WriteLine("Minimul din tablou este {0}", MIN);
            Console.WriteLine("Maximul din tablou este {0}", MAX);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Această problemă determină în paralel atât valoarea minimă dintr-un vector, cât şi cea maximă.

Mecanismul de determinare a valorii minime : se ia o variabilă *MIN* care este iniţializată cu prima valoare din tablou. Se parcurge apoi tabloul de la a doua poziţie până la ultima şi se compară fiecare valoare cu cea din *MIN*. De câte ori se întâlneşte o valoare mai mică decât *MIN*, acea valoare se atribuie lui *MIN*. Astfel, la final, variabila *MIN* va conţine cea mai mică valoare din tablou, deci valoarea minimă.

Mecanismul de determinare a valorii maxime : se ia o variabilă *MAX* care este iniţializată cu prima valoare din tablou. Se parcurge apoi tabloul de la a doua poziţie până la ultima şi se compară fiecare valoare cu cea din *MAX*. De câte ori se întâlneşte o valoare mai mare decât *MAX*, acea valoare se atribuie lui *MAX*. Astfel, la final, variabila *MAX* va conţine cea mai mare valoare din tablou, deci valoarea maximă.

28. *Să se determine primii n termeni ai* ***şirului lui Fibonacci****.*

Istoric : Cunoscut şi ca Leonardo din Pisa, Fibonacci a trăit în secolul XIII şi este considerat a fi unul din cei mai talentaţi matematicieni din Evul Mediu. Unii consideră că Fibonacci este cel care a înlocuit sistemul de cifre romane cu cele arabe.

Şirul lui Fibonacci este o secvenţă recursivă de numere, în care fiecare număr se obţine din suma precedentelor două din şir. Primele două valori se dau şi sunt 1 şi 1. Secvenţa numerelor lui Fibonacci a fascinat de-a lungul istoriei pe foarte mulţi oameni de ştiinţă, matematicieni, fizicieni, biologi, şi continuă să o facă chiar şi în prezent.

Formula de recurenţă: F0=1, F1=1, $F_i=F_{i-1}+F_{i-2}$, i>=2

Program

```
using System;
namespace ConsoleApplication28 {
    class Program
    {
        static void Main()
        {
            int i,n,fn,fn_1,fn_2;
            Console.Write ("Dati nr. de termeni ai sirului: ");
            n = int.Parse(Console.ReadLine());
            fn_1=1; fn_2=1;
            Console.WriteLine("Fibo(0)=1");
            Console.WriteLine("Fibo(1)=1");
            for (i = 2; i < n; i++)
            {
                fn = fn_1 + fn_2;
                fn_2 = fn_1;
                fn_1 = fn;
                Console.WriteLine("Fibo({0})={1}", i, fn);
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Acest program dezvoltă formula de recurenţă descrisă mai sus. Astfel, primii doi termeni, *fn_1* şi *fn_2* sunt iniţializaţi cu 1. Apoi se intră în ciclul *for* care va calcula termenii şirului de la 2 până la *n*. Deoarece această rezolvare nu foloseşte tablouri, tot calculul se realizează cu ajutorul variabilelor *fn_1*, *fn_2* şi *fn*.

Variabila *fn* reprezintă termenul de ordinul *i*, iar *fn_1* şi *fn_2* sunt predecesorii săi de gradul 1 şi 2. Pentru următorul pas, *fn_2* devine vechiul *fn_1*, iar *fn_1* devine *fn*.

Transformarea unui număr din baza de numeraţie 10 într-o altă bază *b*: numărul în baza 10 se împarte la *b* până când se obţine câtul 0. În acel moment toate resturile obţinute în urma acestor împărţiri, luate de la ultimul până la primul, vor reprezenta numărul în baza *b*.

Ex : $55_{10}=110111_2$

55:2=27:2=13:2=6:2=3:2=1:2=0

54	26	12	6	2	0
1	1	1	0	1	1

Transformarea unui număr dintr-o bază *b* în baza 10: Se iau toate cifrele numărului în baza *b* începând cu cea mai din dreapta (considerată poziţia 0), se înmulţesc cu $b^{poziţia}$ şi se adună toate aceste valori. Ceea ce se obţine reprezintă numărul în baza 10.

Ex : $110111_2=55_{10}$

$1*2^5+1*2^4+0*2^3+1*2^2+1*2^1+1*2^0=32+16+4+2+1=55$

Ex : $123_8=83_{10}$

$1*8^2+2*8^1+3*8^0=64+16+3=83$

Baze de numeraţie >=11. O bază *b* din intervalul [2, 10] utilizează cifre din intervalul [0, b-1]. Baza 10 utilizează toate cifrele de la 0 la 9, iar alte cifre nu mai există. Astfel, pentru a reprezenta valori în baze de numeraţie mai mari decât 10 sunt necesare şi alte simboluri. Aceste simboluri vor fi litere ale alfabetului, începând cu litera A. În felul acesta, baza 16 care este cea mai adesea folosită din bazele mai mari decât 10, pe lângă cifrele de la 0 la 9 va mai folosi şi literele alfabetului, de la A la F. Uzual spus, "cifra" 10 va fi reprezentată de litera A,..., "cifra" 15 va fi reprezentată de litera F.

Trecerea din baza 10 într-o bază >=10 se face după acelaşi mecanism enunţat mai sus.

Similar, trecerea dintr-o bază >=10 în baza 10 se face după acelaşi mecanism enunţat mai sus.

Ex : $702_{10}=2BE_{16}$

702:16=43:16=2:16=0

688	32	0
14	11	2

Ex : $2BE_{16}=702_{10}$

$2*16^2+B*16^1+E*16^0=2*16^2+11*16^1+14*16^0=512+176+14=702$

29. *Să se transforme un număr natural din baza 10 în baza 2.*

Ex: $25_{10}=11001_2$

Program

```
using System;

namespace ConsoleApplication29
{
    class Program
    {
        static void Main()
        {
            int nr, cifre=0, i;
            int[] cifbin = new int[20];
            Console.Write ("Dati un număr în baza 10: ");
            nr = int.Parse(Console.ReadLine());
            while (nr!=0)
            {
                cifbin[cifre]=nr%2;
                nr=nr/2;
                cifre++;
            }
            Console.Write ("Reprezentarea în baza 2 a nr: ",nr);
```

```
            for (i=cifre-1; i>=0; i--)
                Console.Write (cifbin[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform teoriei enunţate mai sus, pentru a obţine numărul din baza 10 în baza 2, *nr* se împarte la 2 până când se ajunge la 0. Fiecare rest obţinut se adaugă în tabloul *cifbin*. La final, acest tablou va conţine cifrele în baza 2 ale numărului dat, şi se va tipări de la capăt spre început, tot conform teoriei care spune că numărul în baza *b* reprezintă resturile obţinute în urma împărţirilor repetate la *b*, scrise de la ultimul spre primul.

30. *Să se transforme un număr natural din baza 10 în baza 16.*

Program

```
using System;
namespace ConsoleApplication30 {
    class Program
    {
        static void Main()
        {
            int nr, cifre=0, c=1, i;
            char [] cifhex = new char [20];
            Console.Write ("Dati un număr în baza 10: ");
            nr=int.Parse(Console.ReadLine());
            while (nr!=0)
            {
                c=nr%16;
                if (c <= 9)
                    cifhex[cifre] = (char)(c + 48);
                else
                    cifhex[cifre] = (char)(c + 55);
                nr=nr/16;
                cifre++;
            }
            Console.Write ("Reprezentarea în baza 16 este: ");
            for (i=cifre-1; i>=0; i--)
                Console.Write (cifhex[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform teoriei enunţate mai sus, pentru a obţine numărul din baza 10 în baza 16, *nr* se împarte la 16 până când se ajunge la 0. Fiecare rest obţinut se adaugă în tabloul *cifhex*. Trecerea în baza 16 reprezintă un caz mai special, deoarece intervin şi “cifre” formate din litere. În acest sens, pentru a memora resturile obţinute nu se mai poate folosi un tablou de *int*, ci unul de *char*. Dat fiind acest fapt, numerele obţinute ca şi resturi în urma împărţirilor trebuiesc transformate în caractere.

Caracterele sunt privite din punctul de vedere al codului lor Unicode. De exemplu, codul Unicode al cifrelor 0-9 se află în intervalul 48-57. Astfel, pentru a transforma cifra 0 în caracterul ’0’ (care are codul Unicode 48), va trebui să adunăm 48. Dacă se vor obţine resturile 10-15, acestea vor trebui transformate în caracterele ‘A’-‘F’. Codul Unicode al literelor ‘A’-‘F’ se află în intervalul 65-70. Astfel, pentru a transforma numărul 10 în caracterul ’A’ (care are codul Unicode 65), va trebui să adunăm 55.

La final, acest tablou va conţine cifrele în baza 16 ale numărului dat, şi se va tipări de la capăt spre început, tot conform teoriei care spune că numărul în baza *b* reprezintă resturile obţinute în urma împărţirilor repetate la *b*, scrise de la ultimul spre primul.

Observaţie: În C# există şi o variantă mult mai simplă de a obţine reprezentarea în baza 16 a unei valori întregi şi anume folosind funcţia *ToString* cu parametrul ”X” astfel:

```
int nr;
Console.Write ("Dati un număr în baza 10: ");
nr=int.Parse(Console.ReadLine());
Console.Write ("Numarul in baza 16: "+nr.ToString("X"));
```

31. *Să se transforme un număr natural din baza 10 în baza b* $\in$*[2, 9].*

Program

```
using System;
namespace ConsoleApplication31 {
    class Program {
        static void Main()
        {
            int nr, cifre=0, i, b;
            int[] cifbaza=new int[20];
            Console.Write ("Dati un număr în baza 10: ");
            nr=int.Parse(Console.ReadLine());
            Console.Write ("Dati baza de conversie(2-9): ");
            b=int.Parse(Console.ReadLine());
            while (nr!=0)
            {
                cifbaza[cifre]=nr%b;
                nr=nr/b;
                cifre++;
            }
            Console.Write ("Reprez. în baza {0} a nr. este: ", b);
            for (i=cifre-1; i>=0; i--)
                Console.Write (cifbaza[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform teoriei enunţate mai sus, pentru a obţine numărul din baza 10 în baza *b*, *nr* se împarte la *b* până când se ajunge la 0. Fiecare rest obţinut se adaugă în tabloul *cifbaza*. La final, acest tablou va conţine cifrele în baza *b* ale numărului dat, şi se va tipări de la capăt spre început, tot conform teoriei care spune că numărul în baza *b* reprezintă resturile obţinute în urma împărţirilor repetate la *b*, scrise de la ultimul spre primul.

32. *Să se transforme un număr natural din baza 2 în baza 10.*

Program

```
using System;
namespace ConsoleApplication32
{
    class Program
    {
        static void Main()
        {
            int cifre, p2=1, nb10=0, i, c;
            Console.Write ("Câte cifre are numărul în baza 2: ");
            cifre=int.Parse(Console.ReadLine());
            Console.WriteLine  ("Dati  pe  rand  cifrele  numărului  în  baza  2 
incepand din dreapta: ");
            for (i=1; i<=cifre; i++)
            {
                Console.Write ("Cifra {0} : ",i);
                c=int.Parse(Console.ReadLine());
                nb10+=c*p2;
                p2=p2*2;
            }
            Console.WriteLine  ("Reprez.  în  baza  10  a  numărului  {0}  este  ", 
nb10);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se citesc pe rând de la tastatură toate cifrele numărului în baza 2, începând cu cea mai din dreapta (considerată poziţia 0), se înmulţesc cu $2^{poziţia}$ şi se adună toate aceste valori. Ceea ce se obţine reprezintă numărul în baza 10.

33. *Să se transforme un număr natural din baza 16 în baza 10.*

```
using System;
namespace ConsoleApplication33 {
    class Program
    {
        static void Main()
        {
            int cifre, p16 = 1, nb10 = 0;
            string număr;
           Console.Write("Dati cifrele de la dreapta la stanga:");
            număr = Console.ReadLine();
            Console.Write("Reprez. in baza 10 a nr. este: ");
            foreach (char cb16 in număr)
            {
                if (((char)cb16 >= 65) && ((char)cb16 <= 70))
                    nb10 += (cb16 - 55) * p16;

                if (((char)cb16 >= 48) && ((char)cb16 <= 57))
                    nb10 += (cb16 - 48) * p16;
                p16 = p16 * 16;
            }
```

```
            Console.Write(nb10);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se citesc pe rând de la tastatură toate cifrele numărului în baza 16, începând cu cea mai din dreapta (considerată poziţia 0), însă se citesc ca şi caractere datorită posibilelor litere care pot apărea. Pentru a putea face calculul mai departe, aceste caractere trebuiesc transformate în numerele 0-15. Dacă este vorba de o cifră, înseamnă că are codul Unicode în intervalul 48-57, deci pentru a obţine cifra corespunzătoare va trebui să scădem 48. Dacă este vorba de o literă, înseamnă că are codul Unicode în intervalul 65-70, deci pentru a obţine numărul corespunzător va trebui să scădem 55. Numerele astfel obţinute se înmulţesc cu $16^{\text{poziţia}}$ şi se adună toate aceste valori. Ceea ce se obţine reprezintă numărul în baza 10.

34. *Să se transforme un număr natural din baza b∈[2, 9] în baza 10.*

Program

```
using System;

namespace ConsoleApplication34
{
    class Program
    {
        static void Main()
        {
            int b, cifre, pb=1, nb10=0, i, c;
            Console.Write ("Dati baza (2-9): ");
            b=int.Parse(Console.ReadLine());
            Console.Write ("Câte cifre are nr în baza {0} ? ", b);
            cifre=int.Parse(Console.ReadLine());
            Console.WriteLine  ("Dati  pe  rand  cifrele  numărului  în  baza  {0}
incepand din dreapta: ", b);
            for (i=1; i<=cifre; i++)
            {
                Console.Write ("Cifra {0} : ",i);
                c=int.Parse(Console.ReadLine());
                nb10+=c*pb;
                pb=pb*b;
            }
            Console.WriteLine ("Reprez. în baza 10 a numărului este {0}", nb10);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se citesc pe rând de la tastatură toate cifrele numărului în baza *b,* începând cu cea mai din dreapta (considerată poziţia 0), se înmulţesc cu $b^{\text{poziţia}}$ (care se calculează progresiv în variabila *pb*) şi se adună toate aceste valori. Ceea ce se obţine este stocat în variabila *nb10* şi reprezintă numărul în baza 10.

35. *Fiind dat un vector de numere întregi, să se determine cea mai lungă secvenţă de numere consecutive aflate în ordine crescătoare.*

Ex: 1 3 6 9 5 4 **2 4 6 8 11 17** 15 → cea mai lungă secvenţă crescătoare este: 2 4 6 8 11 17

Program

```
using System;

namespace ConsoleApplication35
{
    class Program {
        static void Main(string[] args) {
            int n, i, lung=0, lung_max, poz_start, poz_max;
            Console.Write("Cate elemente are vectorul ? ");
            n = int.Parse(Console.ReadLine());
            int[] a = new int[n];
            Console.WriteLine("Dati elementele vectorului :");
            for (i = 0; i < n; i++)
            {
                Console.Write("a[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.Write("Vectorul: ");
            for (i = 0; i < n; i++)
                Console.Write("{0} ", a[i]);
            lung_max = 0;
            poz_max=0;
            for (i = 0; i < n-1; i++)
            {
                poz_start = i;
                lung = 1;
                while (i<n-1)
                {
                    if (a[i] < a[i + 1])
                    {
                        lung++;
                        i++;
                    }
                    else break;
                }
                if (lung > lung_max)
                {
                    lung_max = lung;
                    poz_max = poz_start;
                }
            }
            Console.WriteLine();
            Console.Write("Cea mai lunga secv crescatoare ");
            Console.WriteLine  ("se  afla  intre  pozitiile  {0}-{1}",  poz_max,
poz_max+lung_max-1);
            Console.Write ("si are {0} elemente: ",lung_max);
            for (i = poz_max; i < poz_max+lung_max; i++)
                Console.Write("{0} ", a[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

În vederea rezolvării problemei, se folosesc următoarele variabile de tip *int*:

- *lung* care reprezintă lungimea secvenţei curente
- *lung_max* care reprezintă cea mai lungă secvenţă şi care e posibil să fie actualizată de mai multe ori pe parcursul traversării vectorului
- *poz_start* care reprezintă poziţia din vector la care începe secvenţa curentă
- *poz_max* care reprezintă poziţia din vector la care începe cea mai lungă secvenţă

Paşii rezolvării sunt următorii:

- după ce se citeşte vectorul de la tastatură, începem să îl traversăm cu ajutorul unei instrucţiuni *for* începând cu poziţia 0
- variabila *poz_start* este iniţializată cu poziţia curentă din vector, iar *lung* este iniţializată cu 1
- în cadrul unei instrucţiuni *while*, atâta timp cât nu se ajunge la sfârşitul vectorului, iar valorile sunt în ordine crescătoare, se merge mai departe în vector şi se incrementează variabila *lung*
- în momentul în care ordinea crescătoare nu se mai păstrează, se compară lungimea secvenţei găsite (*lung*) cu cea mai lungă secvenţă de până atunci (*lung_max*) şi dacă este cazul, se actualizează *lung_max* şi *poz_max*
- la final se afişează elementele din vector care fac parte din cea mai lungă secvenţă crescătoare.

36. *Se dă un vector de numere întregi (pozitive şi negative). Să se transforme vectorul astfel încât toate valorile negative să fie plasate la început, iar valorile pozitive dupa ele.*

Ex: 2 3 -1 5 -3 -4 7 9 10 -5 → 2 3 10 5 9 7 -4 -3 -1 -5

Program

```
using System;
namespace ConsoleApplication36
{
    class Program
    {
        static void Main(string[] args)
        {
            int n, i, j, aux;
            Console.Write ("Dati n: ");
            n=int.Parse(Console.ReadLine());
            int[] a=new int[n+1];
            for (i = 1; i <= n; i++)
                a[i] = int.Parse(Console.ReadLine());
            i = 1;
            j = n;
            while (i<j)
            {
                if ((a[i] > 0) && (a[j] > 0))
                    j--;
                if ((a[i] < 0) && (a[j] < 0))
                    i++;
                if ((a[i] < 0) && (a[j] > 0))
                {
                    i++;
                    j--;
                }
                if ((a[i] > 0) && (a[j] < 0))
```

```
                {
                    aux = a[i];
                    a[i] = a[j];
                    a[j] = aux;
                    i++;
                    j--;
                }
            }
            for (i = 1; i <= n; i++)
                Console.Write("{0}  ", a[i]);
            Console.ReadLine();
        }
    }
}
```

Analiza programului

Se va parcurge vectorul dinspre ambele capete cu ajutorul a doua contoare i şi j (i creste, j scade), atâta timp cât i<j. Dacă se găseşte o pereche (pozitiv, negativ) valorile se schimbă între ele şi ambele contoare se modifică. Celelalte cazuri posibile presupun diverse modificări ale celor două contoare: dacă a[i]>0 si a[j]>0 atunci j--; dacă a[i]<0 şi a[j]<0 atunci i++; dacă a[i]<0 si a[j]>0 atunci i++, j--. La final se afişează vectorul rezultat.

37. *Fiind dat polinomul de gradul n cu coeficienţi reali* $P(X)=a_nX^n+a_{n-1}X^{n-1}+...+a_1X+a_0$*, să se calculeze P(x), unde x este un număr real dat.*

Ex: P(X)=3X^2-X+1, x=2, P(2)=11

Program

```
using System;

namespace ConsoleApplication37
{
    class Program
    {
        static void Main()
        {
            int n, i;
            float x, p = 0;
            string semn="";
            float[] c = new float[20];
            Console.Write("Dati gradul polinomului: ");
            n = int.Parse(Console.ReadLine());
            for (i = n; i >= 0; i--)
            {
                Console.Write("coeficient pt x^{0}=", i);
                c[i] = float.Parse(Console.ReadLine());
            }
            Console.Write("P(X)= ");
            for (i = n; i >= 1; i--)
                if (c[i] != 0)
                {
                    if (c[i] >= 0 && i < n)
                        semn = "+";
                    if (c[i] == 1)
                        Console.Write(semn + "X^{0}", i);
```

```
                else if (c[i] == -1)
                    Console.Write(semn + "-X^{0}", i);
                else
                    Console.Write(semn + "{0}*X^{1}",c[i],i);
                semn = "";
            }
        if (c[0] > 0)
            Console.Write("+{0}", c[0]);
        else if (c[0] < 0)
            Console.Write(c[0]);
        Console.WriteLine();

        Console.Write("Introduceti valoarea lui x=");
        x = float.Parse(Console.ReadLine());
        for (i = 0; i <= n; i++)
            p = p + c[i] * (float)Math.Pow(x,i);
        Console.WriteLine("P({0}) = {1}", x, p);
        Console.ReadKey();
    }
  }
}
```

Analiza programului

După citirea gradului polinomului în variabila *n*, se citesc coeficienţii polinomului în tabloul *c*. Valorile se pun în tablou de la poziţia *n* până la poziţia 0, pentru a respecta forma naturală a coeficienţilor unui polinom (se începe de la gradul cel mai mare).

După aceasta se citeşte valoarea lui *x* şi se poate merge mai departe la calcularea lui P(x). Pentru a face aceasta, ca şi la matematică, se înlocuieşte peste tot necunoscuta X cu valoarea concretă *x* şi se înmulţeşte cu coeficienţii polinomului. Mai trebuiesc făcute şi ridicări la putere, iar pentru aceasta s-a optat pentru funcţia predefinită în C#, *Math.Pow*. Aceasta primeşte ca şi parametrii baza şi exponentul. Se vor calcula pe rând perechi a_iX^i şi se vor însuma.

Se observă că pentru afişarea polinomului s-au folosit câteva elemente care să permită o afişare cât mai apropiată de varianta matematică. În cazul în care un coeficient este 0, termenul corespunzător din polinom nu mai este tipărit.

Se putea opta şi pentru a construi progresiv puterile lui *x* şi a nu mai utiliza funcţia *Math.Pow*.

Pentru aceasta, bucata de cod:

```
for (i=n; i>=0; i--)
  p = p + c[i] * (float)Math.Pow(x, i);
```

se înlocuia cu:

```
int px=1;
for (i=0; i<=n; i++) {
  p = p + c[i] * px;
  px *= x;
}
```

Observaţie: Pentru *Math.Pow* s-a folosit operatorul de conversie explicită (float), deoarece această metodă returnează *double* şi ar fi fost un conflict între tipurile de date, *p* fiind de tip *float*.

38. *Fiind dat polinomul de gradul n cu coeficienţi reali P(X), să se calculeze P(x)(x-b), unde b este un număr real dat.*

Program

```
using System;
namespace ConsoleApplication38 {
    class Program {
        static void Main() {
            int n, i, b;
            string semn = "";
            Console.Write("Dati gradul polinomului: ");
            n = int.Parse(Console.ReadLine());
            int[] a = new int[n+1];
            int[] ab = new int[n+2];
            Console.WriteLine("Introduceti coeficientii lui X");
            for (i = n; i >= 0; i--) {
                Console.Write("Coeficient pt x^{0}=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.Write("P(X)= ");
            for (i = n; i >= 1; i--)
                if (a[i] != 0)
                {
                    if (a[i] >= 0 && i < n)
                        semn = "+";
                    if (a[i]==1)
                        Console.Write(semn + "X^{0}", i);
                    else if (a[i]==-1)
                        Console.Write(semn + "-X^{0}", i);
                    else
                        Console.Write(semn+"{0}*X^{1}", a[i], i);
                    semn = "";
                }
            if (a[0] > 0)
                Console.Write("+{0}", a[0]);
            else if (a[0] < 0)
                Console.Write(a[0]);
            Console.WriteLine();
            Console.Write("Dati valoarea pentru b: ");
            b = int.Parse(Console.ReadLine());
            ab[n + 1] = a[n];
            for (i = n; i >= 1; i--)
                ab[i] = a[i - 1] - b * a[i];
            ab[0] = -b * a[0];
            Console.Write("P(X)(X-{0})=", b);
            for (i = n + 1; i >= 1; i--)
                if (ab[i] != 0)
                {
                    if (ab[i] >= 0 && i <= n)
                        semn = "+";
                    if (ab[i] == 1)
                        Console.Write(semn + "X^{0}", i);
                    else if (ab[i] == -1)
                        Console.Write(semn + "-X^{0}", i);
                    else
                        Console.Write(semn+"{0}*X^{1}", ab[i], i);
                    semn = "";
                }
```

```
            if (ab[0] > 0)
                Console.Write("+{0}", ab[0]);
            else if (ab[0] < 0)
                Console.Write(ab[0]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

$P(x)(x-b)=x(a_nX^n+a_{n-1}X^{n-1}+...+a_1X+a_0)-b(a_nX^n+a_{n-1}X^{n-1}+...+a_1X+a_0)= a_nX^{n+1}+a_{n-1} X^n+...+a_1X^2+a_0X-ba_nX^n-ba_{n-1}X^{n-1}+...-ba_1X-ba_0 = a_nX^{n+1}+(a_{n-1}-ba_n) X^n+...+(a_1-ba_2)X^2+(a_0-ba_1)X-ba_0=$

$= a_nX^{n+1}+\sum_{i=1}^{n}(a_{i-1}-ba_i)X^i -ba_0$

Se observă că noul polinom care se va obţine în urma produsului are gradul *n+1*, iar coeficienţii săi sunt cei deduşi mai sus. Coeficienţii de la 1 la n se vor calcula după formula: coeficientul termenului de grad *i* este $a_{i-1}-ba_i$. Coeficientul de grad *n+1* este a_n, iar coeficientul de grad 0 este $-ba_0$.

Pentru afişarea polinomului s-au folosit câteva elemente care să permită o afişare cât mai apropiată de varianta matematică. În cazul în care un coeficient este 0, termenul corespunzător din polinom nu mai este tipărit.

39. *Să se calculeze derivata de ordin întâi a unui polinom de grad n cu coeficienţi întregi, reprezentat cu ajutorul unui vector.*

Program

```
using System;
namespace ConsoleApplication39 {
    class Program {
        static void Main()
        {
            int n, i;
            string semn = "";
            Console.Write("Dati gradul polinomului: ");
            n = int.Parse(Console.ReadLine());
            int[] a = new int[n+1];
            int[] ad = new int[n];
            Console.WriteLine("Introduceti coeficientii lui x");
            for (i = n; i >= 0; i--)
            {
                Console.Write("Coeficientul pt x^{0}=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.Write("P(X)=");
            for (i = n; i >= 0; i--)
                if (a[i] != 0)
                {
                    if (a[i] >= 0 && i < n)
                        semn = "+";
                    if (i != 0)
                        Console.Write(semn+" {0}*X^{1}", a[i], i);
                    else
                        Console.Write(semn + " " + a[i]);
                    semn = "";
                }
```

```
            Console.WriteLine();
            for (i = n; i >= 1; i--)
                ad[i - 1] = a[i] * (i);
            Console.Write("P'(X)=");
            for (i = n - 1; i >= 0; i--)
                if (ad[i] != 0)
                {
                    if (ad[i] >= 0 && i < n - 1) semn = "+";
                    if (i != 0)
                        Console.Write(semn+" {0}*X^{1}",ad[i],i);
                    else
                        Console.Write(semn + " " + ad[i]);
                    semn = "";
                }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Dacă $P(X)=a_nX^n+a_{n-1}X^{n-1}+...+a_1X+a_0$, atunci

$P'(X)= na_nX^{n-1}+(n-1)a_{n-1}X^{n-2}+...+a_1$

Polinomul care va reprezenta derivata de ordin întâi a lui P(X) are gradul *n-1*, iar coeficienţii săi se calculează după formula: coeficientul de grad *i* este egal cu coeficientul de grad *i+1* din polinomul iniţial, înmulţit cu *i+1*.

Pentru afişarea polinomului s-au folosit câteva elemente care să permită o afişare cât mai apropiată de varianta matematică. În cazul în care un coeficient este 0, termenul corespunzător din polinom nu mai este tipărit.

40. *Să se adauge un element în interiorul unui vector de numere reale, fără a suprascrie elementele deja existente.*

Program

```
using System;

namespace ConsoleApplication40
{
    class Program
    {
        static void Main()
        {
            int n, i, poz, v;
            Console.Write("Dati dimensiunea tabloului: ");
            n = int.Parse(Console.ReadLine());
            float[] a = new float[n + 1];
            Console.WriteLine("Dati elementele tabloului:");
            for (i = 0; i < n; i++)
            {
                Console.Write("A[{0}]=", i + 1);
                a[i] = float.Parse(Console.ReadLine());
            }
            Console.Write("Pozitia pe care se face inserarea (1-{0} ): ",n);
            poz = int.Parse(Console.ReadLine());
            poz--;
            if ((poz >= 0) && (poz < n))  // test validitate
```

```
            {
                Console.Write("Dati valoarea de inserat: ");
                v = int.Parse(Console.ReadLine());
                for (i = n - 1; i >= poz; i--)
                    a[i + 1] = a[i];
                a[poz] = v;
                Console.WriteLine("Vectorul după inserare:");
                for (i = 0; i <= n; i++)
                    Console.Write("{0} ", a[i]);
            }
            else
                Console.Write("Pozitie inexistenta!");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Pentru a insera un element pe poziţia *poz* într-un vector care deja conţine elemente, este necesar ca prima dată să se elibereze poziţia *poz* pentru a nu se scrie peste elementul deja existent acolo. Deoarece capătul din stânga al vectorului este fix, elementele de la poziţia *poz* până la capăt vor trebui mutate cu o poziţie spre dreapta. Deplasarea teoretic se poate face de la *poz* la *n* sau de la *n* la *poz*. Practic însă, deplasarea de la *poz* la *n* nu funcţionează corect, deoarece dacă se scrie elementul de pe poziţia *poz* pe poziţia *poz+1*, atunci elementul care era pe poziţia *poz+1* se pierde. În schimb, dacă se începe de la poziţia *n*, acest element este mutat pe poziţia *n+1* unde înainte nu exista nimic, iar pe mai departe, elementul de pe poziţia *n-1* este mutat pe poziţia *n*, însă acum nu mai este o problemă, deoarece elementul de pe poziţia *n* a fost deja scris pe poziţia *n+1*, ş.a.m.d. În momentul când toate elementele de la poziţia *poz* până la *n* au fost deplasate spre dreapta cu o poziţie, se poate citi noua valoare care se va plasa în vector pe poziţia *poz*.

Înainte de inserare este important să se facă un test pentru a verifica dacă poziţia introdusă de la tastatură este validă.

41. *Să se şteargă un element din interiorul unui vector de numere reale şi să se acopere spaţiul rămas gol prin deplasarea spre stânga a tuturor elementelor din dreapta sa.*

Program

```
using System;
namespace ConsoleApplication41 {
    class Program {
        static void Main()
        {
            int n, i, poz;
            Console.Write("Dati dimensiunea tabloului: ");
            n = int.Parse(Console.ReadLine());
            float[] a = new float[n];
            Console.WriteLine("Introduceti elementele:");
            for (i = 0; i < n; i++)
            {
                Console.Write("A[{0}]=", i + 1);
                a[i] = float.Parse(Console.ReadLine());
            }
            Console.Write("Pozitia de pe care se face stergerea (1-{0}): ",n);
            poz = int.Parse(Console.ReadLine());
```

```
            poz--;
            if ((poz >= 0) && (poz < n))  // test validitate
            {
                for (i = poz + 1; i < n; i++)
                    a[i - 1] = a[i];
                n--;
                Console.Write("Vectorul după stergere: ");
                for (i = 0; i < n; i++)
                    Console.Write("{0} ", a[i]);
            }
            else
                Console.Write("Pozitie inexistenta!");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

În momentul când se şterge un element de pe o poziţie *poz* dintr-un vector, rămâne un loc gol care ar trebui acoperit. Pentru a-l acoperi se vor deplasa toate elementele de la poziţia *poz+1* până la *n* cu o poziţie spre stânga. De observat şi faptul că în urma ştergerii unui element, vectorul va avea *n-1* elemente.

Înainte de ştergere este important să se facă un test pentru a verifica dacă poziţia introdusă de la tastatură este validă.

42. *Să se simuleze* ***ciurul lui Eratostene*** *(algoritm cu ajutorul căruia pot fi determinate numerele prime mai mici decât un număr dat) intr-un vector de n numere naturale.*

Istoric: Geograful şi astronomul grec Eratostene din Cirena a avut ideea de a transforma proprietatea numerelor prime de a nu fi multiplii nici unuia din numerele mai mici decât ele, într-un criteriu de selecţie (cernere) astfel: din şirul primelor n numere naturale se elimină pe rând multiplii lui 2, 3, 5 etc., elementele rămase fiind cele prime. Astfel, 2 fiind prim, din şir se elimină multiplii lui 2 adică 4, 6, 8 etc. Următorul număr rămas neeliminat la pasul anterior este 3, deci 3 este prim şi se vor elimina numerele 9, 15, 21, etc. (multiplii pari ai lui 3 fiind deja eliminaţi). Şi aşa mai departe, până la determinarea tuturor numerelor prime mai mici decât un număr dat.

Varianta 1 :

Program

```
using System;
namespace ConsoleApplication42 {
    class Program {
        static void Main()
        {
            int n, i, j;
            Console.Write ("Dati n: ");
            n=int.Parse(Console.ReadLine());
            int[] c = new int[n+1];
            for (i=2; i<=n; i++)
                c[i]=i;
            for (i=2; i<=Math.Sqrt(n); i++)
                if (c[i]!=0)
                {
                    j=2;
                    while (i*j<=n)
```

```
                {
                    c[i*j]=0;
                    j++;
                }
            }
        for (i=2; i<=n; i++)
            if (c[i]!=0)
                Console.Write ("{0} ", c[i]);
        Console.ReadKey();
    }
  }
}
```

Analiza programului

Această variantă de simulare a *ciurului lui Eratostene* cu ajutorul unui vector presupune plasarea în vector a tuturor valorilor de la 2 la *n*. După aceasta se va trece la "cernerea" valorilor care nu sunt prime. "Cernerea" se face astfel: se parcurg toate valorile de la 2 la $\sqrt{n}$ şi se elimină toţi multiplii lor din vector. Eliminarea constă de fapt în setarea pe 0 a valorii acelor multiplii. Ceea ce rămâne în final în tablou şi este diferit de 0, reprezintă toate numerele prime mai mici decât *n*, care se vor tipări şi pe ecran.

De ce se merge doar până la $\sqrt{n}$? Deoarece în momentul în care s-a ajuns la $\sqrt{n}$, toate valorile de la $\sqrt{n}$ până la *n* au fost deja eliminate sau sunt numere prime.

Demonstraţie: Orice valoare x>$\sqrt{n}$, dacă nu este număr prim, înseamnă că are cel puţin 2 divizori. Se ştie însă că niciunul din divizori nu poate fi <$\sqrt{n}$, deoarece deja toţi multiplii valorilor <$\sqrt{n}$ au fost deja eliminaţi. Ar rămâne varianta în care ambii divizori sunt >$\sqrt{n}$, însă deoarece $\sqrt{n}*\sqrt{n}$=n, produsul oricăror două valori >$\sqrt{n}$ va da o valoare >n. Deci nu există doi divizori ai lui x, ambii mai mari decât $\sqrt{n}$. Rezultă de aici că orice valoare mai mare decât $\sqrt{n}$ neeliminată este sigur număr prim.

Varianta 2 :

Program

```
using System;

namespace ConsoleApplication42b
{
    class Program
    {
        static void Main()
        {
            int n, i, j;
            Console.Write ("Dati n: ");
            n=int.Parse(Console.ReadLine());
            int[] c = new int [n+1];
            for (i=2; i<=n; i++)
                c[i]=1;
            for (i=2; i<=Math.Sqrt(n); i++)
                if (c[i]!=0)
                {
                    j=2;
```

```
                    while (i*j<=n)
                    {
                        c[i*j]=0;
                        j++;
                    }
                }
            for (i=2; i<=n; i++)
                if (c[i]!=0) Console.Write ("{0}  ", i);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Această variantă de simulare a *ciurului lui Eratostene* cu ajutorul unui vector presupune plasarea în vector pe toate poziţiile, a valorii 1. "Cernerea" valorilor care nu sunt prime se face de data aceasta pe baza indicelui de poziţie în tablou a elementelor (ex: indicele 2 este considerat ca fiind numărul 2). Se face în felul acesta o anume economie în memorarea de valori în tablou.

"Cernerea" se face astfel: se parcurg toate poziţiile de la 2 la $\sqrt{n}$ şi se setează toţi multiplii lor din vector pe 0. Indicii elementelor rămase în final în tablou şi diferite de 0, reprezintă toate numerele prime mai mici decât *n*, care se vor tipări şi pe ecran.

43. *Să se verifice dacă un vector de întregi citit de la tastatură îndeplineşte condiţia de mulţime (nu are valori duplicate). În caz contrar, să se transforme în mulţime (se vor şterge valorile duplicate).*
Exemplu: Vectorul : 1 2 3 1 4 5 1 2 6 2 7 3 8 9 nu este mulţime.
În urma transformării el va deveni : 1 2 3 4 5 6 7 8 9

Program

```
using System;
namespace ConsoleApplication43 {
    class Program
    {
        static void Main(string[] args)
        {
            int n, i, j, k;
            Console.Write("Cate elemente are vectorul ? ");
            n=int.Parse(Console.ReadLine());
            int[] a=new int[n];
            Console.WriteLine("Dati elementele vectorului :");
            for (i = 0; i < n; i++)
            {
                Console.Write("a[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.Write("Vectorul: ");
            for (i = 0; i < n; i++)
                Console.Write("{0} ", a[i]);
            for (i = 0; i < n-1; i++)
                for (j=i+1; j<n; j++)
                    if (a[i] == a[j])
```

```
                {
                    if (j != n - 1)
                        for (k = j + 1; k < n; k++)
                            a[k - 1] = a[k];
                    n--;
                    j--;
                }
            Console.WriteLine();
            Console.Write("Vectorul multime: ");
            for (i = 0; i < n; i++)
                Console.Write("{0} ", a[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Testul de mulţime se face astfel: se parcurge vectorul cu un contor *i*. Oricând se ajunge la o poziţie *i*, cu ajutorul unei a doua instrucţiuni *for* se parcurge bucata de vector [*i+1*, *n*]. În cazul în care *a[i]=a[j]* înseamnă ca s-a găsit o pereche de valori duplicate şi cea de pe poziţia *j* este ştearsă din vector conform algoritmului de la problema 41. După ce va fi parcurs întregul vector, vom avea toate valorile duplicate eliminate, deci vectorul îndeplineşte condiţia de mulţime şi este afişat.

44. *Se dau două mulţimi de numere întregi memorate cu ajutorul vectorilor. Să se calculeze reuniunea celor două mulţimi.*

Program

```
using System;

namespace ConsoleApplication44
{
    class Program
    {
        static void Main()
        {
            int n, m, i, j, poz, găsit;
            Console.Write("Numărul de elem ale primei multimi: ");
            n = int.Parse(Console.ReadLine());
            Console.Write("Numărul de elem ale celei de-a doua multimi: ");
            m = int.Parse(Console.ReadLine());
            int[] a = new int[n+m];
            int[] b = new int[m];
            Console.WriteLine("Dati elementele multimii 1:");
            for (i = 0; i < n; i++)
            {
                Console.Write("a[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.WriteLine("Dati elementele multimii 2:");
            for (i = 0; i < m; i++)
            {
                Console.Write("b[{0}]=", i);
                b[i] = int.Parse(Console.ReadLine());
            }
            poz = n;
```

```
            for (j = 0; j < m; j++)
            {
                găsit = 0;
                for (i = 0; i < n; i++)
                    if (b[j] == a[i])
                    {
                        găsit = 1;
                        break;
                    }
                if (găsit == 0)
                {
                    a[poz] = b[j];
                    poz++;
                }
            }
            Array.Sort(a);
            Console.WriteLine("Rezultatul reuniunii: ");
            n = poz;
            for (i = 0; i < n; i++)
                Console.Write("{0}  ", a[i]);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform cunoştinţelor de la matematică se ştie că reuniunea a două mulţimi înseamnă o nouă mulţime care reprezintă punerea în comun a valorilor celor două mulţimi. Se ştie de asemenea că într-o mulţime nu pot apărea valori duplicate. Programul de mai sus foloseşte doi vectori *a* şi *b* pentru reprezentarea mulţimilor.

Reuniunea se realizează astfel: se adaugă la capătul vectorului *a* toate valorile din *b* care încă nu există în *a*. Pentru a face aceasta, fiecare element din vectorul *b* este căutat în vectorul *a*. Dacă este găsit se trece peste el, iar dacă nu este găsit se adaugă la sfârşitul lui *a*.

Deoarece dimensiunea iniţială a lui *a* se va modifica pe cum se adaugă elemente din *b*, pentru a căuta elementele din *b* exclusiv printre cele din *a*, se va reţine într-o variabilă suplimentară *poz* numărul curent de elemente din *a*. Variabila *poz* se va modifică pe parcursul prelucrării, dar căutarea elementelor din *b* în *a* se face tot timpul până la poziţia *n*, care este numărul iniţial de elemente din *a*.

Ex : a: 1 3 4 6; n=4
b: 2 4 5 9; m=4
poz=4

- se caută valoarea 2 în *a*; nu se găseşte → se adaugă la sfârşitul lui *a*; *poz*=5 → a: 1 3 4 6 2
- se caută valoarea 4 în *a*, până la poziţia n=4; este găsită, deci nu se adaugă la *a*
- se caută valoarea 5 în *a*, până la poziţia n=4; nu se găseşte → se adaugă la sfârşitul lui *a*; *poz*=6 → a: 1 3 4 6 2 5
- se caută valoarea 9 în *a*, până la poziţia n=4; nu se găseşte → se adaugă la sfârşitul lui *a*; *poz*=7 → a: 1 3 4 6 2 5 9

Se tipăreşte la final vectorul reuniune *a*, care are acum *poz* elemente.

45. *Să se calculeze intersecţia a două mulţimi de numere reale reprezentate cu ajutorul vectorilor.*

Program

```
using System;
namespace ConsoleApplication45 {
    class Program {
        static void Main()
        {
            int n, m, q, i, j, poz;
            bool mvida = true;
            Console.Write("Numărul de elem ale primei multimi: ");
            n = int.Parse(Console.ReadLine());
            Console.Write("Numărul de elem ale celei de-a doua multimi: ");
            m = int.Parse(Console.ReadLine());
            int[] a = new int[n];
            int[] b = new int[m];
            q = Math.Min(n, m);
            int[] c = new int[q];
            Console.WriteLine("Dati elementele multimii 1");
            for (i = 0; i < n; i++) {
                Console.Write("a[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.WriteLine("Dati elementele multimii 2");
            for (i = 0; i < m; i++) {
                Console.Write("b[{0}]=", i);
                b[i] = int.Parse(Console.ReadLine());
            }
            poz = 0;
            for (i = 0; i < n; i++)
                for (j = 0; j < m; j++)
                    if (a[i] == b[j])
                    {
                        c[poz] = a[i];
                        poz++;
                        mvida = false;
                        break;
                    }
            Console.Write("Rezultatul intersectiei: ");
            if (mvida==false)
                for (i = 0; i < poz; i++)
                    Console.Write("{0} ", c[i]);
            else
                Console.WriteLine("multimea vida");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform cunoştinţelor de la matematică se ştie că intersecţia a două mulţimi înseamnă o nouă mulţime care conţine toate elementele comune ale celor două mulţimi. Se ştie de asemenea că într-o mulţime <u>nu</u> pot apărea valori duplicate.

Programul de mai sus foloseşte doi vectori *a* şi *b* pentru reprezentarea mulţimilor, iar intersecţia se va depune în vectorul *c*.

Intersecţia se realizează astfel: se parcurge vectorul *a* element cu element. Fiecare element din *a* este căutat în vectorul *b*. Dacă este găsit, se adaugă acest element la vectorul *c*, dacă nu, se trece mai departe în *a*. Căutarea se face cu ajutorul a două instrucţiuni *for* imbricate, primul *for* mergând pe elementele vectorului *a*, iar al doilea *for* merge pe elementele vectorului *b*.

Se tipăreşte la final vectorul intersecţie *c*, care are *poz* elemente.

Vectorul *c* va avea cel mult dimensiunea celei mai mici mulţimi, dintre *a* şi *b*. Din acest motiv, numărul de elemente ale lui *c* a fost stabilit la *Math.Min(n, m)*. (Ex: *a* are 5 elemente, *b* are 8 elemente → *c* are maxim 5 elemente).

S-a folosit variabila *mvida* de tip *bool* care dacă mulţimea *c* nu are nici un element are valoarea *false*. Ea este utilă pentru a şti la final dacă avem ce să afişăm.

46. *Să se calculeze diferenţa A/B a două mulţimi A, B de numere reale reprezentate cu ajutorul vectorilor (diferenţa semnifică elementele care sunt în A şi nu sunt şi în B).*

Program

```
using System;
namespace ConsoleApplication46 {
    class Program
    {
        static void Main()
        {
            int n, m, i, j, poz;
            bool mvida = true, gasit;
            Console.Write("Numărul de elem ale primei multimi: ");
            n = int.Parse(Console.ReadLine());
            Console.Write("Numărul de elem ale celei de-a doua multimi: ");
            m = int.Parse(Console.ReadLine());
            int[] a = new int[n];
            int[] b = new int[m];
            int[] c = new int[n];
            Console.WriteLine("Dati elementele multimii 1");
            for (i = 0; i < n; i++)
            {
                Console.Write("a[{0}]=", i);
                a[i] = int.Parse(Console.ReadLine());
            }
            Console.WriteLine("Dati elementele multimii 2");
            for (i = 0; i < m; i++)
            {
                Console.Write("b[{0}]=", i);
                b[i] = int.Parse(Console.ReadLine());
            }
            poz = 0;
            for (i = 0; i < n; i++)
            {
                gasit=false;
                for (j = 0; j < m; j++)
                    if (a[i] == b[j])
                    {
                        gasit = true;
                        break;
                    }
                if (gasit==false)
```

```
                {
                    c[poz] = a[i];
                    mvida=false;
                    poz++;
                }
            }
            Console.Write("A / B= ");
            if (mvida==false)
                for (i = 0; i < poz; i++)
                    Console.Write("{0} ", c[i]);
            else
                Console.Write("multimea vida");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Conform cunoştinţelor de la matematică se ştie că diferenţa a două mulţimi A/B înseamnă o nouă mulţime care conţine toate elementele din A care nu se află şi în B.

Programul de mai sus foloseşte doi vectori *a* şi *b* pentru reprezentarea mulţimilor, iar diferenţa se va depune în vectorul *c*.

Diferenţa se realizează astfel: se parcurge vectorul *a* element cu element. Fiecare element din *a* este căutat în vectorul *b*. Dacă nu este găsit, se adaugă acest element la vectorul *c*, iar dacă da, se trece mai departe în *a*. Căutarea se face cu ajutorul a două instrucţiuni *for* imbricate, primul *for* mergând pe elementele vectorului *a,* iar al doilea *for* merge pe elementele vectorului *b*.

Se tipăreşte la final vectorul diferenţă *c*, care are *poz* elemente. Vectorul *c* a fost definit ca având maxim *n* elemente, atâtea câte are şi vectorul *a*.

S-a folosit variabila *mvida* de tip *bool* care dacă mulţimea *c* nu are nici un element are valoarea *false*. Ea este utilă pentru a şti la final dacă avem ce să afişăm.

6.3 MATRICI

47. *Să se construiască transpusa unei matrice oarecare de elemente reale.*

Program

```
using System;

namespace ConsoleApplication47
{
    class Program
    {
        static void Main()
        {
            int i, j, m, n;
            Console.Write("Dati nr de linii ale matricii m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricii n=");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            int[,] ta = new int[n, m];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Elementele matricii : ");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            for (i = 0; i < n; i++)
                for (j = 0; j < m; j++)
                    ta[i, j] = a[j, i];
            Console.WriteLine("Elementele matricii transpuse: ");
            // Transpusa are n linii şi m coloane
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < m; j++)
                    Console.Write(ta[i, j] + " ");
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Se ştie de la matematică faptul că transpusa unei matrici înseamnă inversarea liniilor cu coloanele. Astfel o matrice de dimensiuni (mxn) va avea transpusa de dimensiuni (nxm).

În acest program, transpusa se construieşte parcurgând toată matricea *a* şi folosind o a doua matrice *ta* în care se depun valorile matricei iniţiale inversat, adică ceea ce era pe linii în cea iniţială se depune pe coloane în a doua: *ta[i][j]=a[j][i]* (linia *j* din *a* se pune pe coloana *j* din *ta*, iar coloana *i* din *a* se pune pe linia *i* în *ta*) şi la final se afişează matricea *ta*.

48. *Să se interschimbe două linii (coloane) dintr-o matrice oarecare de elemente reale.*

Interschimbarea a 2 linii :

Program

```
using System;

namespace ConsoleApplication48
{
    class Program
    {
        static void Main()
        {
            int i, j, m, n, aux, l1, l2;
            Console.Write("Dati nr de linii ale matricii m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricii n=");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Elementele matricii :");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
          Console.Write("Indicele liniei de schimbat (1-{0}):",m);
            l1 = int.Parse(Console.ReadLine()); l1--;
            Console.Write("Indicele liniei cu care se schimba (1-{0}): ", m);
            l2 = int.Parse(Console.ReadLine()); l2--;
            for (i = 0; i < n; i++)
            {
                aux = a[l1, i];
                a[l1, i] = a[l2, i];
                a[l2, i] = aux;
            }
            Console.WriteLine("Matricea după interschimbarea liniilor:”);
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Interschimbarea unui linii a matricei cu o alta presupune parcurgerea celor două linii în paralel şi interschimbarea a câte unei perechi de valori de pe aceeaşi coloană, lucru care se realizează cu “metoda celor trei pahare” prezentată în problema 1. Aici “metoda celor trei pahare” este folosită în bucata de cod:

```
for (i=0; i<n; i++) {
   aux=a[l1][i];
   a[l1][i]=a[l2][i];
   a[l2][i]=aux;
}
```

În felul acesta, elementul de pe linia *l1*, coloana *i* se va schimba cu elementul de pe linia *l2*, coloana *i*, iar *i* merge de la prima până la ultima coloană de pe o linie, practic pe toată linia.

La final se afişează din nou matricea pentru a se observa interschimbarea liniilor.

Pentru un plus de corectitudine, programul ar trebui să testeze validitatea indicilor celor două linii de interschimbat care se citesc de la tastatură.

49. *Să se calculeze valoarea maximă dintr-o matrice oarecare de elemente reale şi să se afişeze toate poziţiile din matrice unde se găseşte aceasta.*

Program

```
using System;
namespace ConsoleApplication49
{
    class Program
    {
        static void Main()
        {
            int m, n, i, j, max;
            Console.Write("Dati nr de linii ale matricei m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricei n=");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Matricea introdusa:");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            max = a[0, 0];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                    if (a[i, j] > max) max = a[i, j];
            Console.WriteLine("Maximul= {0} şi se afla pe pozitiile : ", max);
```

```
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                    if (a[i, j] == max)
                        Console.WriteLine("({0},{1})  ", i+1,j+1);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Rezolvarea acestei probleme seamănă cu cea a găsirii valorii maxime dintr-un vector. Şi aici se ia o variabilă *max* care este iniţializată cu prima valoare din matrice (valoarea de pe prima linie şi prima coloană). Se parcurge apoi toată matricea şi se compară fiecare valoare cu cea din *max*. De câte ori se întâlneşte o valoare mai mare decât *max*, acea valoare se atribuie lui *max*. Astfel, la final, variabila *max* va conţine cea mai mare valoare din matrice, deci valoarea maximă.

Pentru a afişa toate poziţiile din matrice unde se găseşte valoarea maximă trebuie parcursă din nou toată matricea şi fiecare element al ei este comparat cu maximul. Dacă sunt egale, se tipăreşte linia şi coloana acelui element.

50. *Să se calculeze valoarea minimă dintr-o matrice oarecare de elemente reale şi să se afişeze numărul de apariţii a acestei valori în matrice.*

Program

```
using System;
namespace ConsoleApplication50
{
    class Program
    {
        static void Main()
        {
            int m, n, i, j, min, c = 0;
            Console.Write("Dati nr de linii ale matricei : ");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricei :");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Matricea introdusa:");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                   Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            min = a[0, 0];
```

```
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    if (a[i, j] < min)
                    {
                        min = a[i, j];
                        c = 1;
                    }
                    if (a[i, j] == min)
                        c++;
                }
            Console.Write("Minimul={0} şi apare în matrice de {1} ori", min, c);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Rezolvarea acestei probleme seamănă cu cea a găsirii valorii minime dintr-un vector. Şi aici se ia o variabilă *min* care este iniţializată cu prima valoare din matrice (valoarea de pe prima linie şi prima coloană). Se parcurge apoi toată matricea şi se compară fiecare valoare cu cea din *min*. De câte ori se întâlneşte o valoare mai mică decât *min*, acea valoare se atribuie lui *min*. Astfel, la final, variabila *min* va conţine cea mai mică valoare din matrice, deci valoarea minimă.

Pentru a afişa numărul de apariţii al valorii minime în matrice există cel puţin două variante : 1) după aflarea minimului să se mai parcurgă o dată toată matricea şi să se numere de câte ori este întâlnită valoarea minimă sau 2) numărul de apariţii să se calculeze odată cu calculul valorii minime, astfel : iniţial numărul de apariţii *c* este setat pe 1. Dacă este găsită în matrice o valoare egală cu minimul temporar, se incrementează numărul de apariţii. Dacă minimul este modificat, se iniţializează din nou numărul de apariţii cu 1.

51. *Să se calculeze suma elementelor de pe diagonala principală dintr-o matrice pătratică de elemente întregi.*

Program

```
using System;
namespace ConsoleApplication51 {
    class Program
    {
        static void Main()
        {
            int m, i, j, suma = 0;
            Console.Write("Dati nr de linii/coloane ale matricei patratice:");
            m = int.Parse(Console.ReadLine());
            int[,] a = new int[m, m];
            for (i = 0; i < m; i++)
                for (j = 0; j < m; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Matricea introdusa: ");
            for (i = 0; i < m; i++)
```

```
            {
                for (j = 0; j < m; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            for (i = 0; i < m; i++)
                suma = suma + a[i, i];
            Console.Write("Suma elem. de pe diag. principala = {0}", suma);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Toate elementele de pe diagonala principală a unei matrice pătratice au proprietatea că indicele liniei este acelaşi indicele coloanei. Astfel, se vor însuma toate elementele din matrice de forma $a_{i,i}$, unde i merge pe toate liniile (coloanele) matricei.

52. *Să se calculeze suma elementelor de pe diagonala secundara dintr-o matrice pătratică de elemente întregi.*

Program

// declararea, citirea şi tipărirea matricei ca şi la problema 51.

//calcul suma:

```
            for (i = 0; i < m; i++)
                suma = suma + a[i, m - i - 1];
            Console.Write("Suma elementelor de pe diagonala secundara={0}",
suma);
```

Analiza programului

Toate elementele de pe diagonala secundară a unei matrice pătratice de dimensiune *m* au proprietatea că indicele coloanei este egal cu m-indicele liniei pe care se află. Astfel, la modul general, se vor însuma toate elementele din matrice de forma $a_{i,m-i-1}$, unde i merge pe toate liniile (coloanele) matricei.

De exemplu, într-o matrice pătratică de dimensiune 4, elementele de pe diagonala secundară au indicii (dacă numerotarea începe de la 0): (0, 3), (1, 2), (2, 1), (2, 0).

53. *Să se calculeze suma elementelor din triunghiul de deasupra diagonalei principale dintr-o matrice pătratică de elemente întregi.*

Program

// declararea, citirea şi tipărirea matricei ca şi la problema 51.

//calcul suma:

```
            for (i = 0; i < m - 1; i++)
                for (j = i + 1; j < m; j++)
                    suma = suma + a[i,j];
            Console.Write("Suma elem. de deasupra diag principale = {0}", suma);
```

Analiza programului

Elementele de deasupra diagonalei principale a unei matrici pătratice de dimensiune *m* au

proprietatea că se află pe liniile (dacă numerotarea începe de la 0) i∈0..m-2, iar pe aceste linii se află pe coloanele j∈i+1..m-1. În acest sens, sunt necesare două instrucţiuni *for* care să parcurgă liniile şi coloanele corespunzătoare şi să însumeze valorile.

54. *Să se calculeze suma elementelor din triunghiul de sub diagonala principala dintr-o matrice pătratică de elemente întregi.*

Program

// declararea, citirea şi tipărirea matricei ca şi la problema 51.
//calcul suma:

```
for (i = 1; i < m; i++)
    for (j = 0; j < i; j++)
        suma = suma + a[i,j];
Console.Write("Suma elem. de sub diag. principala = {0}", suma);
```

Analiza programului

Elementele de sub diagonala principală a unei matrice pătratice de dimensiune *m* au proprietatea că se află pe liniile (dacă numerotarea începe de la 0) i∈1..m-1, iar pe aceste linii se află pe coloanele j∈0..i-1. În acest sens, sunt necesare două instrucţiuni *for* care să parcurgă liniile şi coloanele corespunzătoare şi să însumeze valorile.

55. *Să se calculeze suma elementelor din triunghiul de deasupra diagonalei secundare dintr-o matrice pătratică de elemente întregi.*

Program

// declararea, citirea şi tipărirea matricei ca şi la problema 51.
//calcul suma:

```
for (i = 0; i < m - 1; i++)
    for (j = 0; j < m - i - 1; j++)
        suma = suma + a[i, j];
Console.Write("Suma elem de deasupra diag. secundare = {0}", suma);
```

Analiza programului

Elementele de deasupra diagonalei secundare a unei matrici pătratice de dimensiune *m* au proprietatea că se află pe liniile (dacă numerotarea începe de la 0) i∈0..m-2, iar pe aceste linii se află pe coloanele j∈0..m-i-2. În acest sens, sunt necesare două instrucţiuni *for* care să parcurgă liniile şi coloanele corespunzătoare şi să însumeze valorile.

56. *Să se calculeze suma elementelor din triunghiul de sub diagonala secundara dintr-o matrice pătratică de elemente întregi.*

Program

// declararea, citirea şi tipărirea matricei ca şi la problema 51.
//calcul suma:

```
for (i = 1; i < m; i++)
    for (j = m - i; j < m; j++)
        suma = suma + a[i, j];
Console.Write("Suma elem de sub diag. secundara = {0}", suma);
```

Analiza programului

Elementele de sub diagonala secundară a unei matrice pătratice de dimensiune *m* au proprietatea că se află pe liniile (dacă numerotarea începe de la 0) i∈1..m-1, iar pe aceste linii se află pe coloanele j∈m-i..m-1. În acest sens, sunt necesare două instrucţiuni *for* care să parcurgă

liniile şi coloanele corespunzătoare şi să însumeze valorile.

57. *Să se verifice care dintre liniile unei matrice oarecare de elemente întregi este simetrică (elementele egal depărtate de capetele liniei sunt egale).*

Program

```
using System;
namespace ConsoleApplication64
{
    class Program
    {
        static void Main()
        {
            int m, n, i, j;
            bool egale;
            Console.Write("Dati nr de linii ale matricei m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricei n=");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Matricea introdusa: ");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            for (i = 0; i < m; i++)
            {
                egale = true;
                for (j = 0; j < n / 2; j++)
                    if (a[i, j] != a[i, n - j - 1])
                        egale = false;
                if (egale == true)
                    Console.WriteLine("Linia {0} este simetrica", i+1);
            }
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Pentru a determina dacă o linie dintr-o matrice este simetrică, se parcurge acea linie până la jumătate şi se compară pe rând fiecare element din prima jumătate a liniei cu simetricul său din cealaltă jumătate (dacă numerotarea începe de la 0 şi e vorba de poziţia j din prima jumătate dintr-un total de n elemente, atunci simetricul său este n-j-1).

Pentru a rezolva problema, se parcurg toate liniile matricei, şi pentru fiecare linie se aplică raţionamentul enunţat mai sus. Se va începe de fiecare dată prin a presupune că linia este simetrică. Dacă cumva în linie este găsită o pereche de valori simetrice care nu sunt egale, concluzia va fi că linia respectivă nu este simetrică.

58. *Să se calculeze produsul dintre o matrice şi un vector de elemente întregi.*

Program

```
using System;
namespace ConsoleApplication58 {
    class Program {
        static void Main()
        {
            int m, n, i, j;
            Console.Write("Dati nr de linii ale matricei m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de coloane ale matricei n=");
            n = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            int[] b = new int[n];
            int[] c = new int[m];
            for (i = 0; i < m; i++)
                for (j = 0; j < n; j++)
                {
                    Console.Write("a[{0},{1}]=", i, j);
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            Console.WriteLine("Dati elementele vectorului:");
            for (i = 0; i < n; i++) {
                Console.Write("b[{0}]=", i);
                b[i] = int.Parse(Console.ReadLine());
            }
            Console.WriteLine("Matricea introdusa:");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                    Console.Write(a[i, j] + " ");
                Console.WriteLine();
            }
            Console.WriteLine("Vectorul introdus: ");
            for (i = 0; i < n; i++)
                Console.WriteLine(b[i] + " ");
            for (i = 0; i < m; i++)
            {
                c[i] = 0;
                for (j = 0; j < n; j++)
                    c[i] += a[i, j] * b[j];
            }
            Console.WriteLine();
            Console.WriteLine("Rezultatul produsului matrice x vector: ");
            for (i = 0; i < m; i++)
                Console.WriteLine(c[i] + " ");
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Produsul dintre o matrice şi un vector se poate face prin înmulţirea vectorului la dreapta sau la stânga matricei. Acest program face înmulţirea la dreapta. Pentru ca înmulţirea să fie posibilă, matricea trebuie să aibă dimensiunea (*mxn*), iar vectorul trebuie să aibă *n* elemente. Rezultatul înmulţirii va fi un vector cu *m* elemente.

În program este folosită matricea *a* de dimensiune (*mxn*) care urmează a fi înmulţită cu vectorul *b* de dimensiune (*n*), iar rezultatul se va depune în vectorul *c* de dimensiune (*m*). Vectorul *c* se calculează după formula: $c_i=a_{i,j}*b_j$, $j\in0..n-1$, iar $i\in0..m-1$. Se observă că în program au fost necesare două instrucţiuni *for*, pentru a parcurge cu *i* liniile matricei şi cu *j* coloanele. Pentru fiecare linie *i* din matrice va exista câte un c_i. La final se afişează vectorul rezultat *c*.

59. *Să se calculeze produsul a două matrice de elemente întregi de forma (m,n) şi (n,p).*

<u>Cu funcţii:</u>

Program

```
using System;
namespace ConsoleApplication59
{
    class Program
    {
        static void citire(int[,] matrice, int l, int c)
        {
            int i, j;
            for (i = 0; i < l; i++)
                for (j = 0; j < c; j++)
                {
                    Console.Write("[{0},{1}]=", i, j);
                    matrice[i, j] = int.Parse(Console.ReadLine());
                }
        }
        static void tiparire(int[,] matrice, int l, int c)
        {
            int i, j;
            for (i = 0; i < l; i++)
            {
                for (j = 0; j < c; j++)
                    Console.Write(matrice[i, j] + " ");
                Console.WriteLine();
            }
        }
        static void Main()
        {
            int m, n, p, i, j, k;
            Console.Write("Dati nr de linii ale matricei A: m=");
            m = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de col ale matricei A: n=");
            n = int.Parse(Console.ReadLine());
            Console.Write("Dati nr de col. ale matricei B: p=");
            p = int.Parse(Console.ReadLine());
            int[,] a = new int[m, n];
            int[,] b = new int[n, p];
            int[,] c = new int[m, p];
            Console.WriteLine();
            Console.WriteLine("Dati elementele matricei A:");
```

```
            citire(a, m, n);
            Console.WriteLine();
            Console.WriteLine("Dati elementele matricei B:");
            citire(b, n, p);
            Console.WriteLine();
            Console.WriteLine("Matricea A:");
            tiparire(a, m, n);
            Console.WriteLine();
            Console.WriteLine("Matricea B:");
            tiparire(b, n, p);
            Console.WriteLine();
            //produsul
            for (i = 0; i < m; i++)
                for (j = 0; j < p; j++)
                {
                    c[i, j] = 0;
                    for (k = 0; k < n; k++)
                        c[i, j] += a[i, k] * b[k, j];
                }
            Console.WriteLine("Matricea produs A x B:");
            tiparire(c, m, p);
            Console.ReadKey();
        }
    }
}
```

Analiza programului

Pentru ca produsul celor două matrice să fie posibil, este necesar ca prima să aibă dimensiunea (*m*x*n*), iar cea de-a doua dimensiunea (*n*x*p*). Cu alte cuvinte, numărul de coloane al primei să fie egal cu numărul de coloane al celei de-a doua. Rezultatul înmulţirii este tot o matrice, de dimensiune (*m*x*p*).

În program este folosită matricea *a* de dimensiune (*m*x*n*) care urmează a fi înmulţită cu matricea *b* de dimensiune (*n*x*p*), iar rezultatul se va depune în matricea *c* de dimensiune (*m*x*p*).

Valorile lui *c* se calculează după formula: $c_{i,j}=\sum_{k=0}^{n-1} a_{i,k} * b_{k,j}$, iar $i \in 0..m-1$, iar $j \in 0..n-1$.

Se observă că în program au fost necesare trei instrucţiuni *for*, una pentru a parcurge cu *i* liniile matricei *a*, una pentru a parcurge cu *j* coloanele lui *b* şi una pentru a parcurge cu *k* coloanele lui *a*. La final se afişează rezultatul înmulţirii aflat în matricea *c*.

S-a construit o funcţie pentru citirea unei matrice şi una pentru afişarea unei matrice, pentru a nu repeta de mai multe ori acelaşi cod în funcţia Main().

BIBLIOGRAFIE

1. Jeff Ferguson, Brian Patterson, Jason Beres, Pierre Boutquin, and Meeta Gupta - C# Bible, 2002, Wiley Publishing.
2. Daniel Solis- Illustrated C# 2005, Apress, 2006.
3. Herbert Schildt - C#: A begginer's Guide, McGraw-Hill Company, 2001
4. Faraz Rasheed - Programmers Heaven: C# School, Synchron Data, 2006.
5. Charles Petzold - Programming Microsoft Windows with C# (Microsoft), 2002.
6. Andy Harris - Microsoft C# Programming for the Absolute Beginner, Course Technology, 2002.
7. Michael Mcmillan - Data Structures And Algorithms Using C#, Cambridge University Press, 2007.
8. Jack Purdum Beginning - C# 3.0, Wiley Publishing, Inc., 2007.
9. Carmen Fifor – Introducere în programare. Limbajul C – teorie şi probleme rezolvate, Ed. Univ. "Aurel Vlaicu" Arad, 2007

P.O.O.

1. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Ph.D. Jim Conallen, Kelli A. Houston - Object-Oriented Analysis and Design with Applications, Third Edition, Pearson Education, Inc., 2007.

Software

Visual Studio 2010 Express Editions - http://www.microsoft.com/Express/
Visual Studio Professional - http://msdn.microsoft.com/enus/vstudio/products/aa700831.aspx